# Optimizing Linux for AMD EPYC™ 9005 Series Processors with SUSE Linux Enterprise 15 SP6

SUSE Linux Enterprise 15 SP6

AMD EPYC™ 9005 Series Processors

Yousaf Kaukab, Engineering Manager (SUSE)

Mel Gorman, Senior Kernel Engineer (SUSE)

Tony Jones, Senior Kernel Engineer (SUSE)

Martin Jambor, Tool Chain Developer (SUSE)

Dario Faggioli, Virtualization Specialist (SUSE)

Kim Naru, Engineering Manager (AMD)

SUSE

# Optimizing Linux for AMD EPYC™ 9005 Series Processors with SUSE Linux Enterprise 15 SP6

The document at hand provides an overview of both the AMD EPYC™ 9005 Series Processors based on Zen5 and Zen5c cores. It details how some computational-intensive workloads can be tuned on SUSE Linux Enterprise Server 15 SP6.

**Disclaimer:**

# Contents

# 1 Overview

The AMD EPYC 9005 Series Processor is the 5th generation of the AMD EPYC server class processor family. It is based on the Zen 5 microarchitecture introduced in 2024. AMD EPYC 9005 Series Processors based on Zen5 cores support up to 128 cores (256 threads) whereas AMD EPYC 9005 Series Processors based on Zen5c cores support up to 192 cores (384 threads). Both support 12 memory channels per socket. At the time of writing, 1-socket and 2-socket models are expected to be available from Original Equipment Manufacturers (OEMs) in 2024. This document provides an overview of the AMD EPYC 9005 Series Processors based on Zen5 cores and how computational-intensive workloads can be tuned on SUSE Linux Enterprise Server 15 SP6. Additional details about the AMD EPYC 9005 Series Processors based on Zen5c cores are provided where appropriate.

# 2 AMD EPYC 9005 Series Processor (Zen5 cores) architecture

*Symmetric multiprocessing (SMP)* systems are those that contain two or more physical processing cores. Each core may have two threads if *Symmetric multithreading (SMT)* is enabled, with some resources being shared between SMT siblings. To minimize access latencies, multiple layers of caches are used with each level being larger but with higher access costs. Cores may share different levels of cache which should be considered when tuning for a workload.

Historically, a single socket contained several cores sharing a hierarchy of caches and memory channels and multiple sockets were connected via a memory interconnect. Modern configurations may have multiple dies as a *Multi-Chip Module (MCM)* with one set of interconnects within the socket and a separate interconnect for each socket. This means that some CPUs and memory are faster to access than others depending on the "distance". This should be considered when tuning for *Non-Uniform Memory Architecture (NUMA)* as all memory accesses may not reference local memory incurring a variable access penalty.

The 5th Generation AMD EPYC Processor has an MCM design with up to seventeen dies on each package. From a topology point of view, this is significantly different to the 1st Generation AMD EPYC Processor design. However, it is similar to the 4th Generation AMD EPYC Processor other than the increase in die count. One die is a central IO die through which all off-chip communication passes through. The basic building block of a compute die is an eight-core Core CompleX (CCX) with its own L1-L3 cache hierarchy. Similar to the 4th Generation AMD EPYC Processor, one Core Complex Die (CCD) consists of one CCX connected via an Infinity Link to

the IO die, as opposed to two CCXs used in the 2nd Generation AMD EPYC Processor. This allows direct communication within a CCD instead of using the IO link maintaining reduced communication and memory access latency. A 128-core 5th Generation AMD EPYC Processor socket therefore consists of 16 CCDs consisting of 16 CCXs (containing 8 cores each) or 128 cores in total (256 threads with SMP enabled) with one additional IO die for 17 dies in total. This is a large increase in the core count relative to the 4th Generation AMD EPYC Processor.

Both the 4th and 5th Generation AMD EPYC Processors potentially have a larger L3 cache. In a standard configuration, a 5th Generation AMD EPYC Processor has 32MB L3 cache. Some CPU chips may also include an AMD V-Cache expansion that can triple the size of the L3 cache. This potentially provides a major performance boost to applications as more active data can be stored in low-latency cache. The exact performance impact is variable, but any memory-intensive workload should benefit from having a lower average memory access latency because of a larger cache.

Communication between the chip and memory happens via the IO die. Each CCD has one dedicated Infinity Fabric link to the IO die. The practical consequence of this architecture versus the 1st Generation AMD EPYC Processor is that the topology is simpler. The first generation had separate memory channels per die and links between dies giving two levels of NUMA distance within a single socket and a third distance when communicating between sockets. This meant that a two-socket machine for EPYC had 4 NUMA nodes (3 levels of NUMA distance). The 2nd Generation AMD EPYC Processor has only 2 NUMA nodes (2 levels of NUMA distance) which makes it easier to tune and optimize. The NUMA distances are the same for the 3rd, 4th and 5th Generation AMD EPYC Processors.

The IO die has a total of 12 memory controllers supporting DDR5 *Dual Inline Memory Modules (DIMMs)* with the maximum supported speed expected to be DDR5-6000 at the time of writing. This implies a peak channel bandwidth of 48 GB/sec or 576 GB/sec total throughput across a socket. The exact bandwidth depends on the DIMMs selected, the number of memory channels populated, how cache is used and the efficiency of the application. Where possible, all memory channels should have a DIMM installed to maximize memory bandwidth.

While the topologies and basic layout is similar between the 4th and 5th Generation AMD EPYC Processors, there are several micro-architectural differences. The *Instructions Per Cycle (IPC)* has improved by 17% on average for enterprise and cloud workloads and 37% higher in AI and high performance computing (HPC), although the exact improvement is workload-dependent. The improvements result from a variety of factors including, increased cache bandwidth, a larger

L1d cache, improvements in branch prediction, wider front-end, increased number of Arithmetic Logic Units (ALUs), increased floating point throughput, support for AVX-512 with 512-bit data-path. The degree to which these changes affect performance varies between applications.

Power management on the links is careful to minimize the amount of power required. If the links are idle, the power may be used to boost the frequency of individual cores. Hence, minimizing access is not only important from a memory access latency point of view, but it also has an impact on the speed of individual cores.

There are 128 IO lanes supporting PCIe Gen 5.0 per socket. Lanes can be used as Infinity links, PCI Express links, SATA links (maximum 32 links) or CXL 2.0 links (maximum 64 links). The exact number of PCIe 5.0 and configuration links vary by chip and motherboard. This allows very large IO configurations and a high degree of flexibility, given that either IO bandwidth or the bandwidth between sockets can be optimized, depending on the OEM requirements. The most likely configuration is that the number of PCIe links will be the same for 1- and 2-socket machines, given that some lanes per socket will be used for inter-socket communication. While some links must be used for inter-socket communication, adding a socket does not compromise the number of available IO channels. The exact configuration used depends on the platform.

# 3 AMD EPYC 9005 Series Processor (Zen5 cores) topology

*Figure 1, "AMD EPYC 9005 Series Processors based on Zen5 cores Topology"* below shows the topology of an example two socket machine with a fully populated memory configuration generated by the `lstopo` tool.
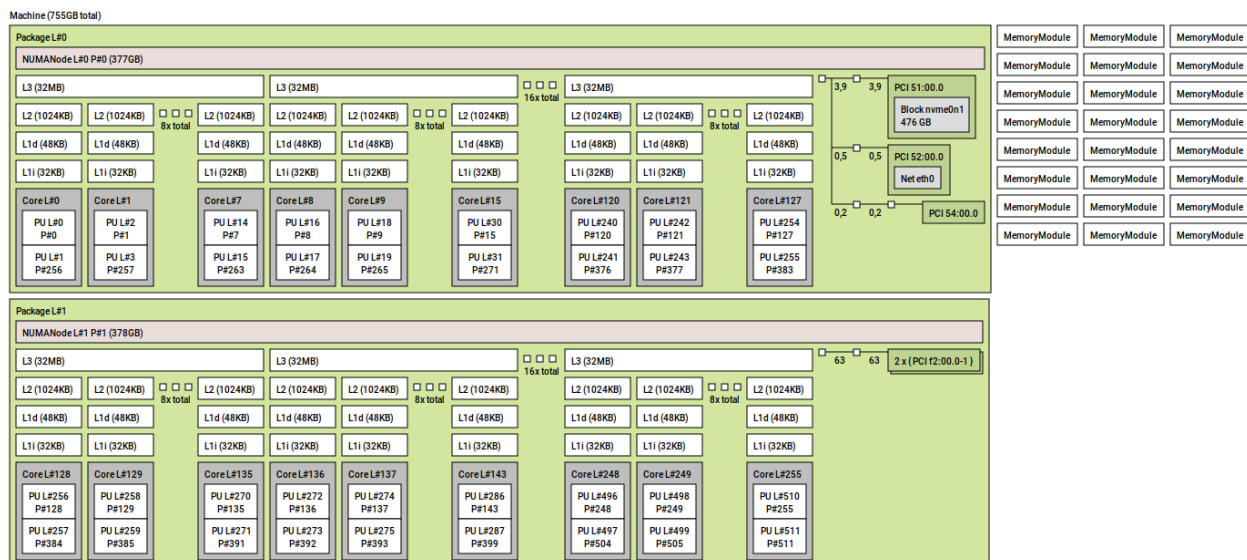
FIGURE 1: AMD EPYC 9005 SERIES PROCESSORS BASED ON ZEN5 CORES TOPOLOGY

This tool is part of the `hwloc-gui` package. The two "packages" correspond to each socket. The CCXs consisting of 8 cores (16 threads) each should be clear, as each CCX has one L3 cache and each socket has 16 CCXs resulting in 128 cores (256 threads). Not obvious are the links to the IO die, but the IO die should be taken into account when splitting a workload to optimize bandwidth to memory. In this example, the IO channels are not heavily used, but the focus will be on CPU and memory-intensive loads. If optimizing for IO, it is recommended that, where possible, the workload is located on the nodes local to the IO channel.

The computer output below shows a conventional view of the topology using the `numactl` tool which is slightly edited for clarity. The CPU IDs that map to each node are reported on the "node X cpus:" lines. They note the NUMA distances on the table at the bottom of the computer output. Node 0 and node 1 are a distance of 32 apart as they are on separate sockets. The distance is not a guarantee of the access latency, it is an estimate of the relative difference. The general interpretation of this distance would suggest that a remote node is 3.2 times longer than a local memory access but the actual latency cost can be different.

```
epyc:~ # numactl --hardware
node 0 cpus: 0 .. 127 256 .. 383
node 0 size: 385996 MB
node 0 free: 384139 MB
node 1 cpus: 128 .. 255 384 .. 511
node 1 size: 386740 MB
node 1 free: 385440 MB
node distances:
node    0    1
```

```
  0:  10  32
  1:  32  10
```

Note that the two sockets displayed are masking some details. There are multiple CCDs and multiple channels meaning that there are slight differences in access latency even to "local" memory. If an application is so sensitive to latency that it needs to be aware of the precise relative distances, then the *Nodes Per Socket (NPS)* value can be adjusted in the BIOS. If adjusted, `numactl` will show additional nodes and the relative distances between them.

Finally, the cache topology can be discovered in a variety of fashions. In addition to `lstopo` which can provide the information, the level, size and ID of CPUs that share cache can be identified from the files under `/sys/devices/system/cpu/cpuN/cache`.

# 4 AMD EPYC 9005 Series Processors (Zen5c cores)

The AMD EPYC 9005 Series Processors based on Zen5c cores launched in 2024. While the fundamental microarchitecture is based on the "Zen 5" compute core, it is optimized for density and efficiency. Its physical layout takes less space and is designed to deliver more performance per watt. There are some other important differences between it and the AMD EPYC 9005 Series Processors based on Zen5 cores. Both processors are socket-compatible, have the same number of memory channels and the same number of I/O lanes. This means that the processors may be used interchangeably on the same platform with the same limitation that dual-socket configurations must use identical processors. Both processors use the same *Instruction Set Architecture (ISA)*. This means that code optimized for one processor will run without modification on the other.

Despite the compatible ISA, the processors are physically different using a manufacturing process focused on increased density for both the CPU core and the physical cache. The L1 and L2 caches have the same capacity. The L3 cache capacity per core is half the capacity of the AMD EPYC 9005 Series Processors based on Zen5 cores as twice as many cores are placed on each CCD. The basic CCX structure in both processors is similar but each CCD with Zen5c has 16 cores instead of 8. While 16 CCDs (with 1 CCX each) with Zen5 cores can be placed in a package, only 12 CCDs with Zen5c cores (each containing 16 cores) can be placed in a package. This increases the maximum number of cores per socket from 128 cores to 192. Finally, the *Thermal Design Points (TDPs)* differ for Zen5c cores, with different frequency scaling limits and generally a lower peak frequency. While each individual Zen5c core may achieve less peak performance than the Zen5 core, the total peak compute throughput available is higher because of the increased number of cores.

The intended use case and workloads determine which processor is superior. The key advantage of the AMD EPYC 9005 Series Processors based on Zen5c cores is packing more cores within the same socket. This may benefit cloud or hyperscale environments in that more containers or virtual machines can use uncontested CPUs for their workloads within the same physical machine. As a result, physical space in data centers can potentially be reduced. It may also benefit some HPC workloads that are primarily CPU and memory bound. For example, some HPC workloads scale to the number of available cores working on data sets that are too large to fit into a typical cache. For such workloads, the AMD EPYC 9005 Series Processors based on Zen5c cores may be ideal.

# 5  Memory and CPU binding

NUMA is a scalable memory architecture for multiprocessor systems that can reduce contention on a memory channel. A full discussion on tuning for NUMA is beyond the scope for this document. But the document "A NUMA API for Linux" at https://halobates.de/numaapi3.pdf ↗ provides a valuable introduction.

The default policy for programs is the "local policy". A program which calls `malloc()` or `mmap()` reserves virtual address space but does not immediately allocate physical memory. The physical memory is allocated the first time the address is accessed by any thread and, if possible, the memory will be local to the accessing CPU. If the mapping is of a file, the first access may have occurred at any time in the past so there are no guarantees about locality.

When memory is allocated to a node, it is less likely to move if a thread changes to a CPU on another node or if multiple programs are remote-accessing the data unless *Automatic NUMA Balancing (NUMAB)* is enabled (see *Section 7, "Automatic NUMA balancing"*). When NUMAB is enabled, unbound process accesses are sampled. If there are enough remote accesses, then the data will be migrated to local memory. This mechanism is not perfect and incurs overhead of its own. This can be important for performance for thread and process migrations between nodes to be minimized and for memory placement to be carefully considered and tuned.

The `taskset` tool is used to set or get the CPU affinity for new or existing processes. An example usage is to confine a new process to CPUs local to one node. Where possible, local memory will be used. But if the total required memory is larger than the node, then remote memory can still be used. In such configurations, it is recommended to size the workload such that it fits in the node. This avoids that any of the data is being paged out when `kswapd` wakes to reclaim memory from the local node.

`numactl` controls both memory and CPU policies for processes that it launches and can modify existing processes. In many respects, the parameters are easier to specify than `taskset`. For example, it can bind a task to all CPUs on a specified node instead of having to specify individual CPUs with `taskset`. Most importantly, it can set the memory allocation policy without requiring application awareness.

Using policies, a preferred node can be specified where the task will use that node if memory is available. This is typically used in combination with binding the task to CPUs on that node. If a workload's memory requirements are larger than a single node and predictable performance is required, then the "interleave" policy will round-robin allocations from allowed nodes. This gives suboptimal but predictable access latencies to main memory. More importantly, interleaving reduces the probability that the operating system (OS) will need to reclaim any data belonging to a large task.

Further improvements can be made to access latencies by binding a workload to a single CCD within a node. As L3 caches are shared within a CCD on the 3rd, 4th and 5th Generation AMD EPYC Processors, binding a workload to a CCD avoids L3 cache misses caused by workload migration. This is an important difference from the 2nd Generation AMD EPYC Processor which favored binding within a CCX.

In most respects, the guidance for optimal bindings for cache and nodes remains the same between the 3rd, 4th and 5th Generation AMD EPYC Processors. However, with SUSE Linux Enterprise 15 SP6, the necessity to bind specifically to the L3 cache for optimal performance is relaxed. The CPU scheduler in SUSE Linux Enterprise 15 SP6 has superior knowledge of the cache topology of all generations of the AMD EPYC Processors and how to balance load between CPU caches, NUMA nodes and memory channels.

## Note: CPU Scheduler Awareness of Cache Topology

With SUSE Linux Enterprise 15 SP6 having superior knowledge of the CPU cache topology and how to balance load, tuning specifically has a smaller impact to performance for a given workload. This is **not** a limitation of the operating system. It is a side-effect of the baseline performance being improved on AMD EPYC Processors in general.

See examples below on how `taskset` and `numactl` can be used to start commands bound to different CPUs depending on the topology.

```
# Run a command bound to CPU 1
epyc:~ # taskset -c 1 [command]
```

```
# Run a command bound to CPUs belonging to node 0
epyc:~ # taskset -c `cat /sys/devices/system/node/node0/cpulist` [command]

# Run a command bound to CPUs belonging to nodes 0 and 1
epyc:~ # numactl —cpunodebind=0,1 [command]

# Run a command bound to CPUs that share L3 cache with cpu 1
epyc:~ # taskset -c `cat /sys/devices/system/cpu/cpu1/cache/index3/shared_cpu_list`
 [command]
```

## 5.1   Tuning for local access without binding

The ability to use local memory where possible and remote memory if necessary is valuable. But there are cases where it is imperative that local memory always be used. If this is the case, the first priority is to bind the task to that node. If that is not possible, then the command `sysctl vm.zone_reclaim_mode=1` can be used to aggressively reclaim memory if local memory is not available.

### Note: Potential Hazard with `vm.zone_reclaim_mode`

While this option is good from a locality perspective, it can incur high costs because of stalls related to reclaim and the possibility that data from the task will be reclaimed. Treat this option with a high degree of caution and testing.

## 5.2   Hazards with CPU binding

There are three major hazards to consider with CPU binding.

The first to watch for is remote memory nodes being used when the process is not allowed to run on CPUs local to that node. The scenarios when this can occur are outside the scope of this paper. However, a common reason is an IO-bound thread communicating with a kernel IO thread on a remote node bound to the IO controller. In such a setup, the data buffers managed by the application are stored in remote memory incurring an access cost for the IO.

While tasks may be bound to CPUs, the resources they are accessing, such as network or storage devices, may not have interrupts routed locally. `irqbalance` generally makes good decisions. But in cases where the network or IO is extremely high-performance or the application has very

low latency requirements, it may be necessary to disable `irqbalance` using **systemctl**. When that is done, the IRQs for the target device need to be routed manually to CPUs local to the target workload for optimal performance.

The second is that guides about CPU binding tend to focus on binding to a single CPU. This is not always optimal when the task communicates with other threads, as fixed bindings potentially miss an opportunity for the processes to use idle CPUs sharing a common cache. This is particularly true when dispatching IO, be it to disk or a network interface, where a task may benefit from being able to migrate close to the related threads. It also applies to pipeline-based communicating threads for a computational workload. Hence, focus initially on binding to CPUs sharing L3 cache. Then consider whether to bind based on an L1/L2 cache or a single CPU using the primary metric of the workload to establish whether the tuning is appropriate.

The final hazard is similar: if many tasks are bound to a smaller set of CPUs, then the subset of CPUs could be oversaturated even though there is spare CPU capacity available.

## 5.3   cpusets and memory control groups

*cpusets* are ideal when multiple workloads must be isolated on a machine in a predictable fashion. cpusets allow a machine to be partitioned into subsets. These sets may overlap, and in that case they suffer from similar problems as CPU affinities. In the event there is no overlap, they can be switched to "exclusive" mode which treats them completely in isolation with relatively little overhead. Similarly, they are well suited when a primary workload must be protected from interference because of low-priority tasks. In such cases the low-priority tasks can be placed in a cpuset. The caveat with cpusets is that the overhead is higher than using scheduler and memory policies. Ordinarily, the accounting code for cpusets is completely disabled. But when a single cpuset is created, there is a second layer of checks against scheduler and memory policies.

Similarly, `memcg` can be used to limit the amount of memory that can be used by a set of processes. When the limits are exceeded, the memory will be reclaimed by tasks within `memcg` directly without interfering with any other tasks. This is ideal for ensuring there is no inference between two or more sets of tasks. Similar to cpusets, there is some management overhead incurred. This means, if tasks can simply be isolated on a NUMA boundary, then this is preferred from a performance perspective. The major hazard is that, if the limits are exceeded, then the processes directly stall to reclaim the memory which can incur significant latencies.

**Note**

Without `memcg`, when memory gets low, the global reclaim daemon does work in the background and if it reclaims quickly enough, no stalls are incurred. When using `memcg`, observe the `allocstall` counter in `/proc/vmstat` as this can detect early if stalling is a problem.

# 6 High performance storage devices and interrupt affinity

High performance storage devices like *Non-Volatile Memory Express (NVMe)* or *Serial Attached SCSI (SAS)* controller are designed to take advantage of parallel IO submission. These devices typically support a large number of submit and receive queues, which are tied to *MSI-X* interrupts. Ideally, these devices should provide as many MSI-X vectors as there are CPUs in the system. To achieve the best performance, each MSI-X vector should be assigned to an individual CPU.

# 7 Automatic NUMA balancing

Automatic NUMA Balancing (NUMAB) is a feature that identifies and relocates pages that are being accessed remotely for applications that are not NUMA-aware. There are cases where it is impractical or impossible to specify policies. In such cases, the balancing should be sufficient for throughput-sensitive workloads, but on occasion, NUMAB may be considered hazardous as it incurs a cost. Under ideal conditions, an application is NUMA aware and uses memory policies to control what memory is accessed and NUMAB simply ignores such regions. However, even if an application does not use memory policies, it is possible that the application still accesses mostly local memory and NUMAB adds overhead confirming that accesses are local which is an unnecessary cost. For latency-sensitive workloads, the sampling for NUMA balancing may be too unpredictable and would prefer to incur the remote access cost or interleave memory instead of using NUMA. The final corner case where NUMA balancing is a hazard happens when the number of runnable tasks always exceeds the number of CPUs in a single node. In this case, the load balancer (and potentially affine wakes) may pull tasks away from the preferred node as identified by NUMAB resulting in excessive sampling and migrations.

If the workloads can be manually optimized with policies, then consider disabling NUMAB by specifying `numa_balancing=disable` on the kernel command line or via `sysctl kernel.numa_balancing`. The same applies if it is known that the application is mostly accessing local memory.

> ### Note: Changes to Automatic NUMA Balancing
>
> While a disconnect between CPU Scheduler and NUMA Balancing placement decisions still potentially exists in SUSE Linux Enterprise 15 SP6 when the machine is heavily overloaded, the impact is much reduced relative to previous releases for most scenarios. The placement decisions made by the CPU Scheduler and NUMA Balancing are now coupled. Situations where the CPU scheduler and NUMA Balancing make opposing decisions are relatively rare.

# 8 Evaluating workloads

The first and foremost step when evaluating how a workload should be tuned is to establish a primary metric such as latency, throughput, operations per second or elapsed time. When each tuning step is considered or applied, it is critical that the primary metric be examined before conducting any further analysis. This is to avoid intensive focus on a relatively wrong bottleneck. Make sure that the metric is measured multiple times to ensure that the result is reproducible and reliable within reasonable boundaries. When that is established, analyze how the workload is using different system resources to determine what area should be the focus. The focus in this paper is on how CPU and memory is used. But other evaluations may need to consider the IO subsystem, network subsystem, system call interfaces, external libraries, etc. The methodologies that can be employed to conduct this are outside the scope of this paper. But the book "Systems Performance: Enterprise and the Cloud" by Brendan Gregg (see http://www.brendangregg.com/systems-performance-2nd-edition-book.html ↗) is a recommended primer on the subject.

## 8.1 CPU utilization and saturation

Decisions on whether to bind a workload to a subset of CPUs require that the CPU utilization and any saturation risk is known. Both the `ps` and `pidstat` commands can be used to sample the number of threads in a system. Typically, `pidstat` yields more useful information with the

important exception of the run state. A system may have many threads, but if they are idle, they are not contributing to utilization. The `mpstat` command can report the utilization of each CPU in the system.

High utilization of a small subset of CPUs may be indicative of a single-threaded workload that is pushing the CPU to the limits and may indicate a bottleneck. Conversely, low utilization may indicate a task that is not CPU-bound, is idling frequently or is migrating excessively. While each workload is different, load utilization of CPUs may show a workload that can run on a subset of CPUs to reduce latencies because of either migrations or remote accesses. When utilization is high, it is important to determine if the system could be saturated. The `vmstat` tool reports the number of runnable tasks waiting for a CPU in the "r" column where any value over 1 indicates that wakeup latencies may be incurred. While the exact wakeup latency can be calculated using trace points, knowing that there are tasks queued is an important step. If a system is saturated, it may be possible to tune the workload to use fewer threads.

Overall, the initial intent should be to use CPUs from as few NUMA nodes as possible to reduce access latency. However, there are exceptions. The AMD EPYC 9005 Series Processor has a large number of high-speed memory channels to main memory, so consider the workload thread activity. If they are cooperating threads or sharing data, isolate them on as few nodes as possible to minimize cross-node memory accesses. If the threads are completely independent with no shared data, it may be best to isolate them on a subset of CPUs from each node. This is to maximize the number of available memory channels and throughput to main memory. For some computational workloads, it may be possible to use hybrid models such as MPI for parallelization across nodes and OpenMP for threads within nodes.

## Note: Updating tuning for AMD EPYC 9005 Series Processor

It is expected that tuning based on the AMD EPYC 7003 Series Processor will also usually perform optimally on latter series processors including AMD EPYC 9005 series. The main consideration is to account for potential differences in L3 cache sizes because of AMD V-Cache if workloads are tuned specifically for cache size. Also, keep in mind that CPU bindings based on caches may potentially be relaxed on SUSE Linux Enterprise 15 SP6.

## 8.2 Transparent Huge Pages

Huge pages are a feature that can improve performance in many cases. This is achieved by reducing the number of page faults, the cost of translating virtual addresses to physical addresses because of fewer layers in the page table and being able to cache translations for a larger portion of memory. *Transparent Huge Pages (THP)* is supported for private anonymous memory that automatically backs mappings with huge pages where anonymous memory could be allocated as `heap`, `malloc()`, `mmap(MAP_ANONYMOUS)`, etc. There is also support for using THP pages backed by `tmpfs` which can be configured at mount time using the `huge=` mount option. While the THP feature has existed for a long time, it has evolved significantly.

Many tuning guides recommend disabling THP because of problems with early implementations. Specifically, when the machine was running for long enough, the use of THP could incur severe latencies and could aggressively reclaim memory in certain circumstances. These problems were resolved by the time SUSE Linux Enterprise Server 15 SP2 was released, and this is still the case for SUSE Linux Enterprise Server 15 SP6. This means there are no good grounds for automatically disabling THP because of severe latency issues without measuring the impact. However, there are exceptions that are worth considering for specific workloads.

Some high-end in-memory databases and other applications aggressively use `mprotect()` to ensure that unprivileged data is never leaked. If these protections are at the base page granularity, then there may be many THP splits and rebuilds that incur overhead. It can be identified if this is a potential problem by using `strace` or `perf trace` to detect the frequency and granularity of the system call. If they are high-frequency, consider disabling THP. It can also be sometimes inferred from observing the `thp_split` and `thp_collapse_alloc` counters in `/proc/vmstat`.

Workloads that sparsely address large mappings may have a higher memory footprint when using THP. This could result in premature reclaim or fallback to remote nodes. An example would be HPC workloads operating on large sparse matrices. If memory usage is much higher than expected, compare memory usage with and without THP to decide if the trade-off is not worthwhile. This may be critical on AMD EPYC 7003, 9004 and 9005 Series Processor given that any spillover will congest the Infinity links and potentially cause cores to run at a lower frequency.

## Note: Sparsely addressed memory

This is specific to sparsely addressed memory. A secondary hint for this case may be that the application primarily uses large mappings with a much higher **Virtual Size** (VSZ, see *Section 8.4, "Memory utilization and saturation"*) than **Resident Set Size** (RSS). Applications which densely address memory benefit from the use of THP by achieving greater bandwidth to memory.

Parallelized workloads that operate on shared buffers with thread counts exceeding the number of available CPUs on a single node may experience a slowdown with THP if the granularity of partitioning is not aligned to the huge page. The problem is that if a large shared buffer is partitioned on a 4K boundary, then false sharing may occur whereby one thread accesses a huge page locally and other threads access it remotely. If this situation is encountered, the granularity of sharing should be increased to the THP size. But if that is not possible, disabling THP is an option.

Applications that are extremely latency-sensitive or must always perform in a deterministic fashion can be hindered by THP. While there are fewer faults, the time for each fault is higher as memory must be allocated and cleared before being visible. The increase in fault times may be in the microsecond granularity. Ensure this is a relevant problem as it typically only applies to extremely latency-sensitive applications. The secondary problem is that a kernel daemon periodically scans a process looking for contiguous regions that can be backed by huge pages. When creating a huge page, there is a window during which that memory cannot be accessed by the application and new mappings cannot be created until the operation is complete. This can be identified as a problem with thread-intensive applications that frequently allocate memory. In this case, consider effectively disabling `khugepaged` by setting a large value in `/sys/kernel/mm/transparent_hugepage/khugepaged/alloc_sleep_millisecs`. This will still allow THP to be used opportunistically while avoiding stalls when calling **malloc()** or **mmap()**.

THP can be disabled. To do so, specify **transparent_hugepage=disable** on the kernel command line, at runtime via `/sys/kernel/mm/transparent_hugepage/enabled` or on a per-process basis by using a wrapper to execute the workload that calls **prctl(PR_SET_THP_DISABLE)**.

## 8.3   User/kernel footprint

Assuming an application is mostly CPU- or memory-bound, it is useful to determine if the footprint is primarily in user space or kernel space. This gives a hint where tuning should be focused. The percentage of CPU time can be measured on a coarse-grained fashion using `vmstat` or a fine-grained fashion using `mpstat`. If an application is mostly spending time in user space, then the focus should be on tuning the application itself. If the application is spending time in the kernel, then it should be determined which subsystem dominates. The `strace` or `perf trace` commands can measure the type, frequency and duration of system calls as they are the primary reasons an application spends time within the kernel. In some cases, an application may be tuned or modified to reduce the frequency and duration of system calls. In other cases, a profile is required to identify which portions of the kernel are most relevant as a target for tuning.

## 8.4   Memory utilization and saturation

The traditional means of measuring memory utilization of a workload is to examine the *Virtual Size (VSZ)* and *Resident Set Size (RSS)*. This can be done by using either the `ps` or `pidstat` tool. This is a reasonable first step but is potentially misleading when shared memory is used and multiple processes are examined. VSZ is simply a measure of memory space reservation and is not necessarily used. RSS may be double accounted if it is a shared segment between multiple processes. The file `/proc/pid/maps` can be used to identify all segments used and whether they are private or shared. The file `/proc/pid/smaps` and `/proc/pid/smaps_rollup` reveals more detailed information including the *Proportional Set Size (PSS)*. PSS is an estimate of RSS except it is divided between the number of processes mapping that segment, which can give a more accurate estimate of utilization. Note that the `smaps` and `smaps_rollup` files are very expensive to read and should not be monitored at a high frequency. This is especially the case if workloads are using large amounts of address space, many threads or both. Finally, the *Working Set Size (WSS)* is the amount of active memory required to complete computations during an arbitrary phase of a program's execution. It is not a value that can be trivially measured. But conceptually it is useful as the interaction between WSS relative to available memory affects memory residency and page fault rates.

On NUMA systems, the first saturation point is a node overflow when the "local" policy is in effect. Given no binding of memory, when a node is filled, a remote node's memory will be used transparently and background reclaim will take place on the local node. Two consequences of

this are that remote access penalties will be used and old memory from the local node will be reclaimed. If the WSS of the application exceeds the size of a local node, then paging and refaults may be incurred.

The first item to identify is whether a remote node overflow occurred, which is accounted for in `/proc/vmstat` as the **numa_hit**, **numa_miss**, **numa_foreign**, **numa_interleave**, **numa_local** and **numa_other counters**:

- **numa_hit** is incremented when an allocation uses the preferred node where preferred may be either a local node or one specified by a memory policy.

- **numa_miss** is incremented when an alternative node is used to satisfy an allocation.

- **numa_foreign** is rarely useful but is accounted against a node that was preferred. It is a subtle distinction from **numa_miss** that is rarely useful.

- **numa_interleave** is incremented when an interleave policy was used to select allowed nodes in a round-robin fashion.

- **numa_local** increments when a local node is used for an allocation regardless of policy.

- **numa_other** is used when a remote node is used for an allocation regardless of policy.

For the local memory policy, the **numa_hit** and **numa_miss** counters are the most important to pay attention to. An application that is allocating memory that starts incrementing the **numa_miss** implies that the first level of saturation has been reached. If monitoring the `proc` is undesirable, then the **numastat** provides the same information. If this is observed on the AMD EPYC 9005 Series Processor, it may be valuable to bind the application to nodes that represent dies on a single socket. If the ratio of hits to misses is close to 1, consider an evaluation of the interleave policy to avoid unnecessary reclaim.

> ✎ Note: NUMA statistics
>
> These NUMA statistics only apply at the time a physical page is allocated and are not related to the reference behavior of the workload. For example, if a task running on node 0 allocates memory local to node 0, then it will be accounted for as a **node_hit** in the statistics. However, if the memory is shared with a task running on node 1, all the accesses may be remote, which is a miss from the perspective of the hardware but not accounted for in `/proc/vmstat`. Detecting remote and local accesses at the hardware level requires using the hardware's *Performance Monitoring Unit* to detect. See perf-mem(1) for further details.

When the first saturation point is reached, reclaim will be active. This can be observed by monitoring the `pgscan_kswapd` and `pgsteal_kswapd` counters in `/proc/vmstat`. If this is matched with an increase in major faults or minor faults, then it may be indicative of severe thrashing. In this case, the interleave policy should be considered. An ideal tuning option is to identify if shared memory is the source of the usage. If this is the case, then interleave the shared memory segments. This can be done in some circumstances using `numactl` or by modifying the application directly.

More severe saturation is observed if the `pgscan_direct` and `pgsteal_direct` counters are also increasing. These counters indicate that the application is stalling while memory is being reclaimed. If the application was bound to individual nodes, increasing the number of available nodes will alleviate the pressure. If the application is unbound, it indicates that the WSS of the workload exceeds all available memory. It can only be alleviated by tuning the application to use less memory or increasing the amount of RAM available.

A more generalized view of resource pressure for CPU, memory and IO can be measured using the kernel *Pressure Stall Information* feature enabled with the command line `psi=1`. When enabled, proc files under `/proc/pressure` show if some or all active tasks were stalled recently contending on a resource. This information is not always available in production. But if the information is available, the memory pressure information may be used to guide whether a deeper analysis is necessary and which resource is the bottleneck.

As before, whether to use memory nodes from one socket or two sockets depends on the application. If the individual processes are independent, either socket can be used. Where possible, keep communicating processes on the same socket to maximize memory throughput while minimizing the socket interconnect traffic.

## 8.5  Other resources

The analysis of other resources is outside the scope of this paper. However, a common scenario is that an application is IO-bound. A superficial check can be made using the `vmstat` tool. This tool checks what percentage of CPU time is spent idle combined with the number of processes that are blocked and the values in the **bi** and **bo** columns. Similarly, if PSI is enabled, then the IO pressure file will show whether some or all active tasks are losing time because of lack of resources. Further analysis is required to determine if an application is IO rather than CPU- or memory-bound. However, this is a sufficient check to start with.

# 9 Power management

Modern CPUs balance power consumption and performance through *Performance States (P-States)*. Low utilization workloads may use lower P-States to conserve power while still achieving acceptable performance. When a CPU is idle, lower power idle states *(C-States)* can be selected to further conserve power. However, this comes with higher exit latencies when lower power states are selected. It is further complicated by the fact that, if individual cores are idle and running at low power, the additional power can be used to boost the performance of active cores. This means this scenario is not a straightforward balance between power consumption and performance. More complexity is added on the AMD EPYC 7003, 9004 and 9005 Series Processors whereby spare power may be used to boost either cores or the Infinity links.

The 5th Generation AMD EPYC Processor is capable of making adjustments to voltage and frequency depending on the historical state of the CPU. There is a latency penalty when switching P-States, but the AMD EPYC 9005 Series Processor is capable of making fine-grained adjustments to reduce the likelihood that the latency is a bottleneck. On SUSE Linux Enterprise Server 15 SP6, cpufreq subsystem uses the `acpi_cpufreq` driver by default for AMD EPYC 9005 Series Processors. However, this may change in the future SUSE Linux Enterprise Server releases as work is in progress to enable `amd-pstate` driver for AMD EPYC 9005 Series Processors. cpufreq subsystem allows P-States to be configured to match requested performance. However, this is limited in terms of the full capabilities of the hardware. It cannot boost the frequency beyond the maximum stated frequencies, and if a target is specified, then the highest frequency below the target will be used. A special case is if the governor is set to **performance**. In this situation the hardware will use the highest available frequency in an attempt to work quickly and then return to idle.

What should be determined is whether power management is likely to be a factor for a workload. A single thread workload that is CPU-bound is likely to run at the highest frequency on a single core. Lastly, a workload that does not communicate heavily with other processes and is mostly CPU-bound is unlikely to experience any side effects because of power management. The exceptions are when load balancing moves tasks away from active CPUs if there is a compute imbalance between NUMA nodes or the machine is heavily overloaded.

The workloads that are most likely to be affected by power management are those that:

- synchronously communicate between multiple threads.

- idle frequently.

- have low CPU utilization overall.

It will be further compounded if the threads are sensitive to wakeup latency.

Power management is critical, not only for power savings, but because power saved from idling inactive cores can be used to boost the performance of active cores. On the other side, low utilization tasks may take longer to complete if the task is not active long enough for the CPU to run at a high frequency. In some cases, problems can be avoided by configuring a workload to use the minimum number of CPUs necessary for its active tasks. Deciding that means monitoring the power state of CPUs.

The P-State and C-State of each CPU can be examined using the **turbostat** utility. The computer output below shows an example, slightly edited to fit the page, where a workload is busy on CPU 0 and other workloads are idle. A useful exercise is to start a workload and monitor the output of **turbostat** paying close attention to CPUs that have moderate utilization and running at a lower frequency. If the workload is latency-sensitive, it is grounds for either minimizing the number of CPUs available to the workload or configuring power management.

```
Pac. Die  Core CPU Avg_M Busy% Bzy_M TSC_M IPC    IRQ    POLL C1   C2    POLL% C1%  C2%
-    -    -    -   5     0.31  1664  2737  1.36   137633 135  3475 99475 0.00  0.11 101.10
0    0    0    0   88    5.38  1638  2696  1.40   1065   0    1    924   0.00  0.00 94.66
0    0    0    256 202   5.79  3494  2696  2.68   1088   0    9    739   0.00  0.03 94.21
0    0    1    1   12    0.73  1622  2696  1.27   493    0    16   344   0.00  0.69 98.59
0    0    1    257 12    0.81  1506  2696  1.22   936    0    5    682   0.00  0.03 99.18
0    0    2    2   9     0.58  1505  2696  1.20   506    0    5    361   0.00  0.04 99.40
0    0    2    258 9     0.59  1507  2696  1.39   672    0    4    489   0.00  0.02 99.41
```

> 📝 Note: turbostat error: Too many open files
>
> Default value of the number of file descriptors that a process may allocate (RLIMIT_NOFILE) is set to 1024 on SLE15-SP6. **turbostat** needs to open 1028 file descriptors on a system with 512 CPUs. Although its possible for an application to increase the current limit, the version of **turbostat** that ships with SLE15-SP6 at the time of writing is not changing this limit. As a result it fails with error *open failed: Too many open files*. A fix for this issue is on the way. Meanwhile, **turbostat** can be run after locally changing this limit by running the command **ulimit -n 1029**.

If tuning CPU frequency management is appropriate, the following actions can be taken to set the management policy to performance using the **cpupower** utility:

```
epyc:~# cpupower frequency-set -g performance
Setting cpu: 0
Setting cpu: 1
```

```
Setting cpu: 2
...
```

Persisting it across reboots can be done via a local **init** script, via **udev** or via one-shot **systemd** service file if necessary. Note that **turbostat** will still show that idling CPUs use a low frequency. The impact of the policy is that the highest P-State will be used as soon as possible when the CPU is active. In some cases, a latency bottleneck will occur because of a CPU exiting idle. If this is identified on the AMD EPYC 9005 Series Processor, restrict the C-state by specifying **processor.max_cstate=2** if lower P-States exist on the kernel command line. This will prevent CPUs from entering lower C-states. The availability of P-states can be determined with **cpupower idle-info**. It is expected on the AMD EPYC 9005 Series Processor that the exit latency from C1 is very low. But by allowing C2, it reduces interference from the idle loop injecting micro-operations into the pipeline and should be the best state overall. It is also possible to set the max idle state on individual cores using **cpupower idle-set**. If SMT is enabled, the idle state should be set on both siblings.

# 10   Security mitigation

On occasion, a security fix is applied to a distribution that has a performance impact. The most notable example is **Meltdown** and multiple variants of **Spectre** but includes others such as ForeShadow (L1TF). The AMD EPYC 9005 Series Processor is immune to the Meltdown variant and page table isolation is never active. However, it is vulnerable to a subset of Spectre variants although **retbleed** is a notable exception. The following table lists all security vulnerabilities that affect the 5th Generation AMD EPYC Processor. In addition, it specifies which mitigations are enabled by default for SUSE Linux Enterprise Server 15 SP6.

TABLE 1: SECURITY MITIGATIONS FOR AMD EPYC 9005 SERIES PROCESSORS

| Vulnerability | Affected | Mitigations |
|---|---|---|
| ITLB Multihit | No | N/A |
| L1TF | No | N/A |
| MDS | No | N/A |
| Meltdown | No | N/A |
| MMIO Stale Data | No | N/A |

| Vulnerability | Affected | Mitigations |
|---|---|---|
| Retbleed | No | N/A |
| Speculative Store Bypass | Yes | prctl and seccomp policy |
| Spectre v1 | Yes | usercopy/swapgs barriers and __user pointer sanitization |
| Spectre v2 | Yes | Retpoline, RSB filling, and conditional IBPB, IBRS_FW, and STIBP |
| SRBDS | No | N/A |
| TSX Async Abort | No | N/A |
| Register File Data Sampling (RFDS) | No | N/A |
| Gather Data Sampling (GDS) | No | N/A |
| Speculative Return Stack Overflow (SRSO) | No | N/A |

If it can be guaranteed that the server is in a trusted environment running only known code that is not malicious, the `mitigations=off` parameter can be specified on the kernel command line. This option disables all security mitigations and may improve performance in some cases. However, at the time of writing and in most cases, the gain on an AMD EPYC 9005 Series Processor is marginal when compared to other CPUs. Evaluate carefully whether the gain is worth the risk and if unsure, leave the mitigations enabled.

# 11   Hardware-based profiling

The AMD EPYC 9005 Series Processor has extensive **Performance Monitoring Unit** (PMU) capabilities. Advanced monitoring of a workload can be conducted via the **perf**. The command supports a range of hardware events including cycles, L1 cache access/misses, TLB access/miss-

es, retired branch instructions and mispredicted branches. To identify what subsystem may be worth tuning in the OS, the most useful invocation is `perf record -a -e cycles sleep 30`. This captures 30 seconds of data for the entire system. You can also call `perf record -e cycles command` to gather a profile of a given workload. Specific information on the OS can be gathered through tracepoints or creating probe points with `perf` or `trace-cmd`. But the details on how to conduct such analyses are beyond the scope of this paper.

# 12 Compiler selection

SUSE Linux Enterprise ships with multiple versions of GCC. SUSE Linux Enterprise 15 SP6 service packs ship with `GCC 7` which at the time of writing is `GCC 7.5.0` with the package version `7-3.9.1`. The intention is to avoid unintended consequences when porting code that may affect the building and operation of applications. The `GCC 7` development originally started in 2016, with a branch created in 2017 and `GCC 7.5` released in 2019. This means that the system compiler has no awareness of the AMD EPYC 7002 or later Series processors.

Fortunately, the add-on `Developer Tools Module` includes additional compilers with the latest version currently based on `GCC 13.2.1`. This compiler is capable of generating optimized code targeted at the 4th Generation AMD EPYC Processor using the `znver4` target. It also provides additional support for `OpenMP 5.0`, extends the support of `OpenMP 5.1` features and very limited first support of `OpenMP 5.2` features. Unlike the system compiler, the major version of GCC shipped with the Developer Tools Module can change during the lifetime of the product. It is expected that `GCC 14` will be included in future releases for generating optimized code for the 5th Generation AMD EPYC Processor. Unfortunately, at the time of writing, there is not a version of GCC available optimized for the AMD EPYC 9005 Series Processor specifically.

The OS packages are built against a generic target. However, where applications and benchmarks can be rebuilt from source, the minimum option should be `-march=znver4` for `GCC 13` and later versions of GCC.

Further information on how to install the `Developer Tools Module` and how to build optimized versions of applications can be found in the guide Advanced optimization and new capabilities of GCC 12 (https://documentation.suse.com/sbp/devel-tools/html/SBP-GCC-12/index.html) ↗.

# 13 Candidate workloads

The workloads that will benefit most from the 5th Generation AMD EPYC Processor architecture are those that can be parallelized and are either memory or IO-bound. This is particularly true for workloads that are "NUMA friendly". They can be parallelized, and each thread can operate independently for most of the workload's lifetime. For memory-bound workloads, the primary benefit will be taking advantage of the high bandwidth available on each channel. For IO-bound workloads, the primary benefit will be realized when there are multiple storage devices, each of which is connected to the node local to a task issuing IO.

## 13.1 Test setup

The following sections will demonstrate how an OpenMP and MPI workload can be configured and tuned on an AMD EPYC 9005 Series Processor reference platform. The system has two processors, each with 128 cores and SMT enabled for a total of 256 cores (512 logical CPUs). The peak bandwidth available to the machine depends on the type of DIMMs installed and how the DIMM slots are populated. Note that the peak theoretical transfer speed is rarely reached in practice, given that it can be affected by the mix of reads/writes and the location and temporal proximity of memory locations accessed.

TABLE 2: TEST SETUP

| | |
|---|---|
| CPU | 2x AMD EPYC 9755 |
| Platform | AMD Engineering Sample Platform |
| Drive | Samsung SSD PM9A1 512GB |
| OS | SUSE Linux Enterprise Server 15 SP6 |
| Memory Type | 24x 32GB DDR5 |
| Memory Interleaving | Channel |
| Memory Speed | 4800 MT/sec |
| Kernel command line | `mitigations=off` |

## 13.2　Test workload: STREAM

*STREAM* is a memory bandwidth benchmark created by Dr. John D. McCalpin from the University of Virginia (for more information, see https://www.cs.virginia.edu/stream/ ↗). It can be used to measure bandwidth of each cache level and bandwidth to main memory while calculating four basic vector operations. Each operation can exhibit different throughputs to main memory depending on the locations and type of access.

The benchmark was configured to run both single-threaded and parallelized with OpenMP to take advantage of multiple memory controllers. The array of elements for the benchmark was set at 536,870,912 elements at compile time so that each array was 4096MB in size for a total memory footprint of approximately 12288 MB. The size was selected in line with the recommendation from STREAM that the array sizes be at least 4 times the total size of L3 cache available in the system. Pay special attention to the exact size of the L3 cache if V-Cache is present. An array-size offset was used so that the separate arrays for each parallelized thread would not share a Transparent Huge Page. The reason is that NUMA balancing may choose to migrate shared pages leading to some distortion of the results.

TABLE 3: TEST WORKLOAD: STREAM

| Compiler | gcc-13 (SUSE Linux) 13.2.1 |
|---|---|
| Compiler flags | `-Ofast -march=znver4 -mcmodel=medium -DOFFSET=512` |
| OpenMP compiler flag | `-fopenmp` |
| OpenMP environment variables | `OMP_PROC_BIND=SPREAD` `OMP_NUM_THREADS=32` |

The `march=znver4` is a reflection of the compiler available in SUSE Linux Enterprise 15 SP6 at the time of writing. It should be checked if a later `GCC` version is available in the `Developer Tools Module` that supported `march=znver5`. The number of OpenMP threads was selected to have at least one thread running for every memory channel by having one thread per L3 cache available. The `OMP_PROC_BIND` parameter spreads the threads such that one thread is bound to each available dedicated L3 cache to maximize available bandwidth. This can be verified using **perf**, as illustrated below with slight editing for formatting and clarity.

```
epyc:~ # perf record -e sched:sched_migrate_task ./stream
```

```
epyc:~ # perf script
...
        stream-nnn x: sched:sched_migrate_task: comm=stream pid=494780 prio=120
 orig_cpu=0 dest_cpu=8
        stream-nnn x: sched:sched_migrate_task: comm=stream pid=494781 prio=120
 orig_cpu=0 dest_cpu=16
        stream-nnn x: sched:sched_migrate_task: comm=stream pid=494782 prio=120
 orig_cpu=0 dest_cpu=24
        stream-nnn x: sched:sched_migrate_task: comm=stream pid=494783 prio=120
 orig_cpu=0 dest_cpu=32
        stream-nnn x: sched:sched_migrate_task: comm=stream pid=494784 prio=120
 orig_cpu=0 dest_cpu=40
        stream-nnn x: sched:sched_migrate_task: comm=stream pid=494785 prio=120
 orig_cpu=0 dest_cpu=48
        stream-nnn x: sched:sched_migrate_task: comm=stream pid=494786 prio=120
 orig_cpu=0 dest_cpu=56
...
```

Several options were considered for the test system that were unnecessary for STREAM running on the AMD EPYC 9005 Series Processor but may be useful in other situations. STREAM performance can be limited if a load/store instruction stalls to fetch data. CPUs may automatically prefetch data based on historical behavior but it is not guaranteed. In limited cases, depending on the CPU and workload, this may be addressed by specifying `-fprefetch-loop-arrays` and depending on whether the workload is store-intensive, `-mprefetchwt1`. However, care must be taken as an explicitly scheduled prefetch may disable a CPU's predictive algorithms and degrade performance. Similarly, for some workloads branch mispredictions can be a major problem, and in some cases breach mispredictions can be offset against I-Cache pressure by specifying `-funroll-loops`. In the case of STREAM on the test system, the CPU accurately predicted the branches rendering the unrolling of loops unnecessary. For math-intensive workloads it can be beneficial to link the application with `-lmvec` depending on the application. In the case of STREAM, the workload did not use significant math-based operations and so this option was not used. Some styles of code blocks and loops can also be optimized to use vectored operations by specifying `-ftree-vectorize` and explicitly adding support for CPU features such as `-mavx2`. In all cases, STREAM does not benefit as its operations are very basic. But it should be considered on an application-by-application basis and when building support libraries such as numerical libraries. In all cases, experimentation is recommended but caution advised. This holds particularly true when considering options like prefetch that may have been advisable on much older CPUs or completely different workloads. Such options are not universally beneficial or always suitable for modern CPUs such as the AMD EPYC 9005 Series Processors.

In the case of STREAM running on the AMD EPYC 9005 Series Processor, it was sufficient to enable `-Ofast`. This includes the `-O3` optimizations to enable vectorization. In addition, it gives some leeway for optimizations that increase the code size with additional options for fast math that may not be standards-compliant.

For OpenMP, the `SPREAD` option was used to spread the load across L3 caches. OpenMP has a variety of different placement options if manually tuning placement. But careful attention should be paid to `OMP_PLACES`, given the importance of the L3 Cache topology in AMD EPYC 9005 Series Processors, if the operating system does not automatically place tasks appropriately. At the time of writing, it is not possible to specify `l3cache` as a place similar to what MPI has. In this case, the topology will need to be examined either with library support such as `hwloc`, directly via the `sysfs` or manually. While it is possible to guess via the CPUID, it is not recommended as CPUs may be offlined or the enumeration may vary between platforms because of BIOS implementations. An example specification of places based on L3 cache for the test system is:

```
{0:8,256:8},   {8:8,264:8},   {16:8,272:8}, {24:8,280:8}, {32:8,288:8}, {40:8,296:8},
{48:8,304:8},  {56:8,312:8},  {64:8,320:8}, {72:8,328:8}, {80:8,336:8},  {88:8,344:8},
{96:8,352:8},  {104:8,360:8}, {112:8,368:8},{120:8,376:8},{128:8,384:8}, {136:8,392:8},
{144:8,400:8}, {152:8,408:8}, {160:8,416:8},{168:8,424:8},{176:8,432:8}, {184:8,440:8},
{192:8,448:8}, {200:8,456:8}, {204:8,464:8},{216:8,472:8},{224:8,480:8}, {232:8,488:8},
{240:8,496:8}, {248:8,504:8}
```

*Figure 2, "STREAM Bandwidth, Single Threaded and Parallelized"* shows the reported bandwidth for the single, parallelized and parallelized with proper placement cases. The single-threaded bandwidth for the basic Copy vector operation on a single core was 51.3 GB/sec. This is higher than the theoretical maximum of a single DIMM, but the IO die interleave accesses, and caching effects and prefetch still apply. The total throughput for each parallel operation with `SPREAD` enabled ranged from 600 GB/sec to 700 GB/sec depending on the type of operation and how efficiently memory bandwidth was used. This is twice as much compared to default placement by OpenMP. This is very roughly scaling with the number of memory channels available on the machine.

## Note: STREAM scores

Higher STREAM scores can be reported by reducing the array sizes so that cache is partially used with the maximum score requiring that each threads memory footprint fits inside the L1 cache. Additionally, it is possible to achieve results closer to the theoretical maximum by manual optimization of the STREAM benchmark using vectored instructions

and explicit scheduling of loads and stores. The purpose of this configuration was to illustrate the impact of properly binding a workload that can be fully parallelized with data-independent threads.
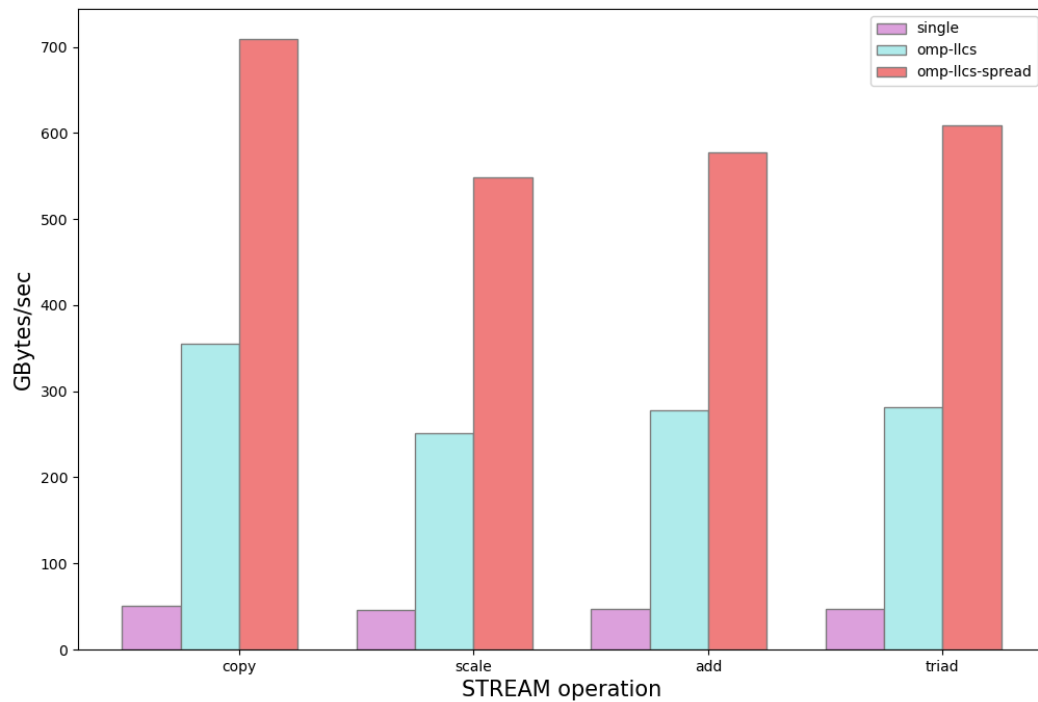


FIGURE 2: **STREAM BANDWIDTH, SINGLE THREADED AND PARALLELIZED**

## 13.3   Test workload: NASA Parallel Benchmark

*NASA Parallel Benchmark (NPB)* is a small set of compute-intensive kernels designed to evaluate the performance of supercomputers. They are small compute kernels derived from *Computational Fluid Dynamics (CFD)* applications. The problem size can be adjusted for different memory sizes. Reference implementations exist for both MPI and OpenMP. This setup will focus on the MPI reference implementation.

While each application behaves differently, one common characteristic is that the workload is very context-switch intensive, barriers frequently yield the CPU to other tasks and the lifetime of individual processes can be very short-lived. The following paragraphs detail the tuning selected for this workload.

The most basic step is setting the CPU governor to "performance" although it is not mandatory. This can address issues with short-lived or mobile tasks failing to run long enough for a higher P-State to be selected even though the workload is very throughput-sensitive. The migration cost parameter is set to reduce the frequency in which the load balancer will move an individual task. The minimum granularity is adjusted to reduce overscheduling effects.

Depending on the computational kernel used, the workload may require a power-of-two number or a square number of processes to be used. However, note that using all available CPUs can mean that the application can contend with itself for CPU time. Furthermore, as IO may be issued to shared memory backed by disk, there are system threads that also need CPU time. Finally, if there is CPU contention, MPI tasks can be stalled waiting on an available CPU and OpenMPI may yield tasks prematurely if it detects there are more MPI tasks than CPUs available. These factors should be carefully considered when tuning for parallelized workloads in general and MPI workloads in particular.

In the specific case of testing NPB on the System Under Test, there was usually a limited advantage to limiting the number of CPUs used. For the *Embarrassingly Parallel (EP)* load in particular, it benefits from using all available CPUs. Hence, the default configuration used all available CPUs (512) which is both a power-of-two and square number of CPUs because it was a sensible starting point. However, this is not universally true. Using `perf`, it was found that some workloads are memory-bound and do not benefit from a high degree of parallelization. In addition, for the final configuration, some workloads were parallelized to have one task per L3 cache in the system to maximize cache usage. The exception was the *Scalar Pentadiagonal (SP)* workload which was both memory-bound and benefited from using all available CPUs. As the number of cores can vary between chips and the number of populated memory channels, the tuning parameters used for this test may not be universally true for all AMD EPYC platforms. This highlights that there is no universal good choice for optimizing a workload for a platform. Thus, experimentation and validation of tuning parameters is vital.

The basic compilation flags simply turned on all safe optimizations. The tuned flags used `-Ofast` which can be unsafe for some mathematical workloads but generated the correct output for NPB. The other flags used the optimal instructions available on the distribution compiler and vectorized some operations. `GCC 13` is more strict in terms of matching types in Fortran. Depending on the version of NPB used, it may be necessary to specify the `-fallow-argument-mismatch` or `-fallow-invalid-boz` to compile unless the source code is modified.

As NPB uses shared files, an XFS partition was used for the temporary files. It is, however, only used for mapping shared files and is not a critical path for the benchmark and no IO tuning is necessary. In some cases, with MPI applications, it will be possible to use a `tmpfs` partition for OpenMPI. This avoids unnecessary IO assuming the increased physical memory usage does not cause the application to be paged out.

TABLE 4: TEST WORKLOAD: NASA PARALLEL BENCHMARK

| | |
|---|---|
| Compiler | gcc-13 (SUSE Linux) 13.2.1 |
| OpenMPI | openmpi4-4.1.6-150600.1.6.x86_64 |
| Default compiler flags | `-m64 -O3 -mcmodel=large` |
| Default number processes | `512` |
| Selective number processes | `bt=256 ep=512 lu=256 mg=256 sp=256` |
| Fortran compiler flags | `-fallow-argument-mismatch -fallow-invalid-boz` |
| Tuned compiler flags | `-Ofast -march=znver4 -mtune=znver4 -ftree-vectorize` |
| CPU governor performance | `cpupower frequency-set -g performance` |
| mpirun parameters | `-mca btl ^openib,udapl -np 512 --bind-to l3cache` |
| mpirun environment | `TMPDIR=/xfs-data-partition` |

*Figure 3, "NAS MPI Results"* shows the time, as reported by the benchmark, for each of the kernels to complete.
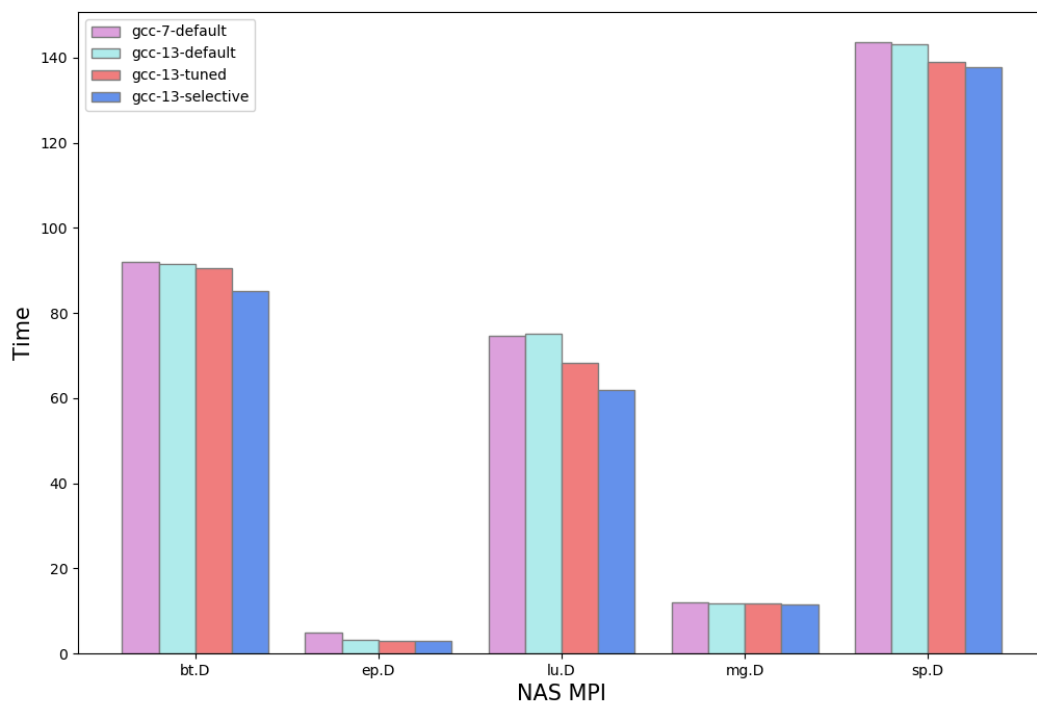
FIGURE 3: NAS MPI RESULTS

The *gcc-7-default* test used the system compiler, all available CPUs, basic compilation options and the `performance` governor. The second test *gcc-13-default* used an alternative compiler. *gcc-13-tuned* used additional compilation options, and bound tasks to L3 caches gaining between 1.5% and 40.6% performance on average relative to *gcc-7-default*. The final test *selective* used processes that either used all CPUs, avoided heavy overloaded or limited processes to one per L3 cache, showing additional between 2.35% and 37.66% depending on the computational kernel.

In some cases, it will be necessary to compile an application that can run on different CPUs. In such cases, `-march=znver4` may not be suitable if it generates binaries that are incompatible with other vendors. In such cases, it is possible to specify the ISA-specific options that are cross-compatible with many x86-based CPUs such as `-mavx2`, `-mfma`, `-msse2` or `msse4a` while favoring optimal code generation for AMD EPYC 9005 Series Processors with `-mtune=znver4`. This can be used to strike a balance between excellent performance on a single CPU and great performance on multiple CPUs.

# 14 Tuning AMD EPYC 9005 Processors (Zen5c cores)

As the Zen5 and Zen5c cores are ISA-compatible, no code tuning or compiler setting changes should be necessary. For Cloud environments, partitioning or any binding of Virtual CPUs to Physical CPUs may need to be adjusted to account for the increased number of cores. The additional cores may also allow additional containers or virtual machines to be hosted on the same physical machine without CPU contention. Similarly, the degree of parallelization for HPC workloads may need to be adjusted. In cases where the workload is tuned based on the number of CCD's, adjustments may be necessary for the changed number of CCDs. An exception are cases where the workload already hits scaling limits. While tuning based on the different number of CCXs is possible, it should only be necessary for applications with very strict latency requirements. As the size of the cache per core is halved, partitioning based on cache sizes may also need to be adjusted. In some cases, where workloads are tuned based on the output of tools like `hwloc` partitioning may adjust automatically but any static partitioning should be re-examined.

When configuring workloads for AMD EPYC 9005 Series Processors based on either Zen5 for Zen5c cores, the most important task is to set expectations. While super-linear scaling is possible, it should not be expected. It may be possible to achieve super-linear scaling in Cloud Environments for the number of instances hosted without performance loss if individual containers or virtual machines are not utilizing 100% of CPU. However, it should be planned carefully and tested. This would be particularly true in cases where multiple instances are hosted that have different times of day or year for active phases. The normal expectation is a best case of 33% gain for CPU-intensive workloads because of the increased number of cores. But sub-linear scaling is common because of resource contention. Contention between SMT siblings, memory bandwidth, memory availability, memory interconnects, thread communication overhead or peripheral devices may prevent perfect linear scaling even for perfectly parallelized applications. Similarly, not all applications can scale perfectly. It is possible for performance to plateau and even regress as the degree of parallelization increases.

# 15 Performance Monitoring Unit changes

Support for the following Zen related perf changes are present in SUSE Linux Enterprise 15 SP6:

1. AMD Performance Monitoring V2 (PerfMonV2) (v5.19) Zen4+

- global registers allowing enabling/disabling of multiple counters concurrently (via a single MSR operation)

- dynamic detection of available PMUs

2. AMD Zen4 IBS extensions (v5.19 kernel, v6.0 perf userspace) Zen4+

- DataSrc extension allowing the source of data to be decoded.
  Perf script and perf record (raw trace) will now report DataSrc details for tagged load/store operations. See perf-amd-ibs(1) for more details.

- IBS L3 miss filtering.
  Enabled by specifying an l3missonly=1 event parameter to perf. See perf-amd-ibs(1) for more details.

3. Last Branch Record Extension v2 (LbrExtV2) (v6.1) Zen4+

- LBR Freeze on PMI

- Hardware branch filtering providing additional specificity

- Branch speculation information

4. JSON event file updates for Zen 4 (v6.2 perf userspace) Zen4+

- This adds event information from the "Core Performance Monitor", "L3 Cache Performance Monitor", "Fabric Performance Monitor Counter" and "Performance Measurement" sections of the AMD Processor Programming Reference (PPR) documentation for Zen4.

Please note. Zen5 JSON event updates (perf userspace) were added in Linux v6.10 and will be supported in a subsequent SLE release. Support for Zen 4 unified memory controller events (kernel and perf userspace) is expected to be available in a similar timeframe.

# 16 Using AMD EPYC 9005 Series Processors for virtualization

Running Virtual Machines (VMs) has some aspects in common with running "regular" tasks on a host operating system. Therefore, most of the tuning advice described so far in this document are valid and applicable to this section too.

However, virtualization workloads do pose their own specific challenges and some special considerations need to be made, to achieve a better tailored and more effective set of tuning advice, for cases where a server is used only as a virtualization host. And this is especially relevant for large systems, such as AMD EPYC 9005 Series Processors.

This is because:

- VMs typically run longer, and consume more memory, than most of others "regular" OS processes.

- VMs can be configured to behave either as NUMA-aware or non NUMA-aware "workloads".

In fact, VMs often run for hours, days, or even months, without being terminated or restarted. Therefore, it is almost never acceptable to pay the price of suboptimal resource partitioning and allocation, even when there is the expectation that things will be better next time. For example, it is always desirable that vCPUs run close to the memory that they are accessing. For reasonably big NUMA aware VMs, this happens only with proper mapping of the virtual NUMA nodes of the guest to physical NUMA nodes on the host. For smaller, NUMA-unaware VMs, that means allocating all their memory on the smallest possible number of host NUMA nodes, and making their vCPUs run on the pCPUs of those nodes as much as possible.

Also, poor mapping of virtual machine resources (virtual CPUs and memory, but also IO) on the host topology induces performance degradation to everything that runs inside the virtual machine – and potentially even to other components of the system .

Regarding NUMA-awareness, a VM is called out to be NUMA aware if a (virtual) NUMA topology is defined and exposed to the VM itself and if the OS that the VM runs (guest OS) is also NUMA aware. On the contrary, a VM is called NUMA-unaware if either no (virtual) NUMA topology is exposed or the guest OS is not NUMA aware.

VMs that are large enough (in terms of amount of memory and number of virtual CPUs) to span multiple host NUMA nodes, benefit from being configured as NUMA aware VMs. However, even for small and NUMA-unaware VMs, wise placement of their memory on the host nodes, and effective mapping of their virtual CPUs (vCPUs) on the host physical CPUs (pCPUs) is key for achieving good and consistent performance.

This second half of the present document focuses on tuning a system where CPU and memory intensive workloads run inside VMs. We will explain how to configure and tune both the host and the VMs, in a way that performance comparable to the ones of the host can be achieved.

Both the Kernel-based Virtual Machine (KVM) and the Xen-Project hypervisor (Xen), as available in SUSE Linux Enterprise Server 15 SP6, provide mechanisms to enact this kind of resource partitioning and allocation. Note that this document focuses on the former (KVM), but it includes some **hints** about how to deal with the latter as well.

> ✎ **Note: Virtual Machine types**
>
> KVM only supports one type of VM – a fully hardware-based virtual machine (HVM). Under Xen, VMs can be paravirtualized (PV) or hardware virtualized machines (HVM).
>
> Xen HVM guests with paravirtualized interfaces enabled (called PVHVM, or HVM) are very similar to KVM VMs, which are based on hardware virtualization but also employ paravirtualized IO (VirtIO). In this document, we always refer to Xen VMs of the (PV)HVM type.

# 17 Resources allocation and tuning of the host

No details are given, here, about how to install and configure a system so that it becomes a suitable virtualization host. For similar instructions, refer to the SUSE documentation at SUSE Linux Enterprise Server 15 SP4 Virtualization Guide: Installation of Virtualization Components (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-vt-installation.html) ↗ .

The same applies to configuring things such as networking and storage, either for the host or for the VMs. For similar instructions, refer to suitable chapters of OS and virtualization documentation and manuals. As an example, to know how to assign network interfaces (or ports) to one or more VMs, for improved network throughput, refer to SUSE Linux Enterprise Server 15 SP6 Virtualization Guide: Assigning a Host PCI Device to a VM Guest (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-libvirt-config-gui.html#sec-libvirt-config-pci) ↗ .

## 17.1 Allocating resources to the host OS

Even if the main purpose of a server is "limited" to running VMs, some activities will be carried out on the host OS. In fact, in both Xen and KVM, the host OS is at least responsible for helping with the IO of the VMs. It may, therefore, be necessary to make sure that the host OS has some resources (namely, CPUs and memory) exclusively assigned to itself.

> ## 📝 Note: Host OS on KVM and on Xen
>
> On KVM, the host OS is the SUSE Linux Enterprise distribution installed on the server, which then loads the hypervisor kernel modules. On Xen, the host OS still is a SUSE Linux Enterprise distribution, but it runs inside what is to all the effect an (although special) Virtual Machine (called Domain 0, or Dom0).

In the absence of any specific requirements involving host resources, a good rule of thumb suggests that 5 to 10 percent of the physical RAM should be reserved to the host OS. On KVM, increase that quota in case the plan is to run many (for example hundreds) of VMs. On Xen, it is okay to always give dom0 not more than a few gigabytes of memory. This is especially the case when planning to take advantage of disaggregation (see https://wiki.xenproject.org/wiki/Dom0_Disaggregation ↗).

In terms of CPUs, depending on the workload, it may be fine to use all the physical CPUs for the VMs (and this is in fact how the benchmarks in the experimental section of this guide have been conducted). On the other hand, if it is necessary to keep some CPUs for the host OS / dom0, we advise to reserve at least one physical core on each NUMA node. This is especially true for a system like the one show in *Figure 1, "AMD EPYC 9005 Series Processors based on Zen5 cores Topology"*. In fact, host OS activities are mostly related to performing IO for VMs and it is beneficial for performance if the kernel threads that handle devices can run on the nodes to which the devices themselves are attached, which is both NUMA nodes, in our case.

System administrators need to be able to reach out and login to the system, to monitor, manage and troubleshoot it. Therefore, it is possible that even more resources needs to be assigned to the host OS. This would be for making sure that management tools (for example, the **SSH** daemon) can be reached, and that the hypervisor's toolstack (for example, `libvirt`) can run without too much contention.

## 17.1.1   Reserving CPUs and memory for the host on KVM

When using KVM, sparing, for example, 32 cores (that is one full core for each CCX on both NUMA nodes) and 64 GB of RAM for the host OS is done by stopping creating VMs when the total number of vCPUs of all VMs has reached 448 (as each core has 2 threads) and when the total cumulative amount of allocated RAM has reached 690 GB.

To make sure that these CPUs are not used by the virtual machines and are available to the host, virtual CPU pinning (discussed later in this document) can be used. There are also other ways to enforce this, for example with `cgroups`, or by means of the `isolcpus` boot parameter, but these are not covered in details within this guide.

## 17.1.2   Reserving CPUs and memory for the host on Xen

When using Xen, dom0 resource allocation needs to be done explicitly at system boot time. For example, assigning 32 physical cores and 64 GB of RAM to it is done by specifying the following additional parameters on the hypervisor boot command line (for example, by editing `/etc/defaults/grub`, and then updating the boot loader):

```
dom0_mem=65536M,max:65536M dom0_max_vcpus=64
```

The number of vCPUs is 64 because we want Dom0 to have 32 physical cores, and the AMD EPYC 9005 Series Processor has Symmetric multithreading (SMT). 65536M memory (that is 64 GB) is specified both as current and as maximum value, to prevent Dom0 from using ballooning (see https://wiki.xenproject.org/wiki/Tuning_Xen_for_Performance#Memory ↗ ).

Making sure that Dom0 vCPUs run on specific pCPUs is not strictly necessary. However, if wanted, it can be enforced acting on the Xen scheduler, when Dom0 is booted (as there is currently no mechanism to set up this via Xen boot time parameters). If using the `xl` toolstack, the command is:

```
xl vcpu-pin 0 <vcpu-ID> <pcpu-list>
```

Or, with `virsh`:

```
virsh vcpupin 0 -\-vcpu <vcpu-ID> -\-cpulist <pcpu-list>
```

Note that **`virsh vcpupin —config ...`** is not effective for Dom0.

If wanting to limit Dom0 to only a specific (set of) NUMA node(s), the **`dom0_nodes=<nodeid>`** boot command line option can be used. This will affect both memory and vCPUs. In fact, it means that memory for Dom0 will be allocated on the specified node(s), and the vCPUs of Dom0 will be restricted to run on those same node(s). When the Dom0 has booted, it is still possible to use **`xl vcpu-pin`** or **`virsh vcpupin`** to change where its vCPUs will be scheduled. But the memory will never be moved from where it has been allocated during boot. On AMD EPYC 9005 Series Processors, using this option is not recommended.

### 17.1.3    Reserving CPUs for the host under IO intensive VM workloads

Tuning the host and the VMs for running IO-intensive workloads is out of the scope of this guide. Just as general advice, if IO done in VM is important, it may be appropriate to leave to the host OS either one core or one thread for each IO device used by each VM. If this is not possible, for example because it reduces to much the CPUs that remain available for running VMs (especially in case the goal is to run many of them), then exploring alternative solutions for handling IO devices (such as *SR-IOV*) is recommended.

## 17.2    (Transparent) Huge Pages

For virtualization workloads, rather than using Transparent Huge Pages (THP) on the host, it is recommended that 1 GB huge pages are used for the memory of the VMs. This sensibly reduces both the page table management overhead and the level of resource contention that the system faces when VMs update their page tables. Moreover, if the host is entirely devoted to running VMs, Huge Pages are likely not required for host OS processes. Actually, in this case, having them active on the host may even negatively affect performance, as the THP service daemon (which is there for merging "regular" pages and form Huge Pages) can interfere with VMs' vCPUs execution. For disabling Huge Pages for the host OS, in a KVM setup, add the following host kernel command-line option:

```
transparent_hugepage=never
```

Or execute this command, at runtime:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

To use 1 GB Huge Pages as backing memory of KVM guests, such pages need to be allocated at the host OS level. It is best to make that happen by adding the following boot time parameter:

```
default_hugepagesz=1GB hugepagesz=1GB hugepages=<number of hugepages>
```

The value `<number of hugepages>` can be computed by taking the amount of memory we want to devoted to VMs, and dividing it by the page size (1 GB). For example on our host with 754 GB of RAM, creating 672 Huge Pages means we can accommodate up to GB of VMs' RAM, and leave plenty (~80GB) to the host OS.

On Xen, none of the above is necessary. In fact, Dom0 is a paravirtualized guest, for which Huge Pages support is not present. On the other hand, for the memory used for the hypervisor and for the memory allocated by the hypervisor for HVM VMs, Huge Pages are always used as much as possible. So no explicit tuning is needed.

## 17.3 Automatic NUMA balancing

On KVM, NUMAB can be useful in dynamic virtualization scenarios, where VMs are created, destroyed and re-created relatively quickly. Or, in general, it can be useful in cases where it is not possible or desirable to statically partition the host resources and assign them explicitly to VMs. This, however, comes at the price of some latency being introduced. Plus, NUMAB's own operation can interfere with VMs' execution and cause further performance degradation. In any case, this document focuses on achieving the best possible performance for VMs, through specific tuning and static resource allocation, therefore it is recommended to leave NUMAB turned off. This can be done by adding the following parameter to the host kernel command line:

```
numa_balancing=disable
```

If anything changes and the system is repurposed to achieve different goals, NUMAB can be enabled on-line with the command:

```
echo 0 > /proc/sys/kernel/numa_balancing
```

On Xen, Automatic NUMA Balancing (NUMAB) in the host OS should be disabled. Dom0 is, currently, a paravirtualized guest without a (virtual) NUMA topology and, thus, NUMAB would be totally useless.

## 17.4 Services, daemons and power management

Service daemons have been discussed already, in the first part of the document. Most of the consideration done there, applies here as well.

For example, `tuned` should either not be used, or the profile should be set to one that does not implicitly put the CPUs in polling mode. Both `throughput-performance` and `virtual-host` profiles from SUSE Linux Enterprise Server 15 SP6 are okay, as neither of them touches `/dev/cpu_dma_latency`.

`irqbalance` can be a source of latency, for no significant performance improvement. The suggestion is again to disable it for latency sensitive workloads. This means, though, that IRQs may need to be manually bound to the appropriate CPUs, considering the IO topology.

As far as power management is concerned, the `cpufreq` governor can either be kept as it is by default, or switched to `performance`, depending on the nature of the workloads of interest.

> **Note: Power management**
>
> For anything that concerns power management on KVM, changing the `tuned` profile or using `cpupower`, from the host OS will have the effects one can expect, and described already in the first half of the document. On Xen, CPU frequency scaling is enacted by the hypervisor. It can be controlled from within Dom0, but the tool that needs to be used is different. For example, for setting governor to `performance`, we need the following:
>
> ```
> xenpm set-scaling-governor performance
> ```

## 17.5 Confidential Computing Technologies (SEV and SEV-ES)

Secure Encrypted Virtualization (SEV) and SEV with Encrypted State (SEV-ES) technologies, are available on the 5th Generation AMD EPYC Processors and supported on SUSE Linux Enterprise Server 15 SP6. They allow the memory of the VMs to be encrypted, enabling a high level of confidentiality. SEV-ES is considered superior to plain SEV, as also CPU registers are encrypted when they are saved into the host memory. For using SEV-ES for a VM, we need to enable it in the VM's own configuration file, but there are preparation steps that needs to occur at the host level.

The `libvirt` documentation contains all the necessary steps required for enabling SEV-ES on a host that supports it. It has been enough to add the following boot parameters to the host kernel command line, in the boot loader:

```
mem_encrypt=on kvm_amd.sev=1
```

For further details, refer to libvirt documentation: Enabling SEV on the host (https://libvirt.org/kbase/launch_security_sev.html#Host) ↗ .

It should be noted that, at least as far as the workload analyzed in this document, enabling SEV-ES on the host has not caused any noticeable performance degradation. In fact, running CPU and memory intensive workloads, both on the host and in VMs, with or without the parameters above, resulted in indistinguishable results. Therefore, enabling SEV-ES at the host level can be considered safe, from the point of view of not hurting performance of VMs and of workloads that will not take advantage of VM memory encryption.

On the other hand, what happens when SEV and SEV-ES are used for encrypting VMs' memory is described later in the guide.

### Note: SEV-SNP

5th Generation AMD EPYC Processors come with an even more advanced confidential computing feature, known as SNP, that is also able to guarantee (among other things) the integrity of the VM. Such feature, however, is not officially available and supported on SUSE Linux Enterprise Server 15 SP6, and is therefore not discussed any further in this document (although it is, actually, already usable via an experimental module).

### Note: Encryption on Xen

In SUSE Linux Enterprise Server 15 SP6, neither SEV nor SEV-ES (not to mention SNP) are available on Xen.

# 18 Resources allocation and tuning of VMs

For instructions on how to create an initial VM configuration, start the VM, and install a guest OS, refer to the SUSE documentation at SUSE Linux Enterprise Server 15 SP6 Virtualization Guide: Guest Installation (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-kvm-inst.html) ↗ .

From a VM configuration perspective, the two most important factors for achieving top performance on CPU and memory bound workloads are:

1. Partitioning of host resources and placement of the VMs

2. Enlightenment of the VMs about their virtual topology

For example, if there are two VMs, each one should be run on one socket, to minimize CPU contention and maximize and memory access parallelism. Also, and especially if the VMs are big enough, they should be made aware of their own virtual topology. That way, all the tuning actions described in the first part of this document become applicable to the workloads running inside the VMs too.

## 18.1 Placement of VMs

When a VM is created, memory is allocated on the host to act as its virtual RAM. Moving this memory, for example on a different NUMA node from the one where it is first allocated, incurs (when possible) in a significant performance impact. Therefore, it is of paramount importance

that the initial placement of the VMs is as optimal as possible. Both Xen and KVM can make "educated guesses" on what a good placement might be. For the purpose of this document, however, we are interested in what is the absolute best possible initial placement of the VMs, taking into account the specific characteristics of AMD EPYC 9005 Series Processor systems. And this can only be achieved by **manually** doing the initial placement. Of course, this comes at the price of reduced flexibility, and is only possible when there is no oversubscription.

Deciding on what pCPUs the vCPUs of a VM run, can be done at creation time, but also easily changed along the VM lifecycle. It is, however, still recommended to start the VMs with good vCPUs placement. This is particularly important on Xen, where vCPU placement is what actually drives and controls memory placement.

Placement of VM memory on host NUMA nodes happens by means of the <numatune> XML element:

```
<numatune>
  <memory mode='strict' nodeset='0-1'/>
  <memnode cellid='0' mode='strict' nodeset='0'/>
  <memnode cellid='1' mode='strict' nodeset='1'/>
  ...
</numatune>
```

The `'strict'` guarantees that the all memory for the VM will come only from the (set of) NUMA node(s) specified in `nodeset`. A cell is, in fact, a virtual NUMA node, with `cellid` being its ID and `nodeset` telling exactly from what host physical NUMA node(s) the memory for this virtual node should be allocated.

The correctness of this kind of tuning can be verified checking on which NUMA node(s) the memory of the QEMU processes representing the VMs have been allocated on, by using the `numastat` tool (on the host) like this:

```
host:~ # numastat -p qemu-system-x86_64
Per-node process memory usage (in MBs) for PID 53153 (qemu-system-x86)
                          Node 0          Node 1           Total
                  --------------- --------------- ---------------
Huge                   344064.00       344064.00       688128.00
Heap                        0.00          150.57          150.57
Stack                       0.00            0.13            0.13
Private                   149.59         4522.84         4672.43
                  --------------- --------------- ---------------
Total                  344213.59       348737.54       692951.12
```

In fact, we see that there is only 1 QEMU process (as there is only 1 VM running) and its memory footprint has been equally split between the two NUMA nodes.

A NUMA-unaware VM can still include this element in its configuration file. It will have only one `<memnode>` element, and the output of `numastat` will look like this:

```
host:~ # numastat -p qemu-system-x86_64
Per-node process memory usage (in MBs)
PID                             Node 0          Node 1            Total
----------------------  --------------- --------------- ---------------
184717 (qemu-system-x86)      346040.98           19.30        346060.28
184980 (qemu-system-x86)          19.50       346227.04        346246.54
----------------------  --------------- --------------- ---------------
Total                         346060.48       346246.34        692306.82
```

In this case there are 2 VMs running and their memory have been pretty much entirely allocated on a single NUMA node: `Node 0` for the VM associated to the QEMU process with PID `184717`, and `Node 1` for the other one.

A similar, but more complex example is also shown below. There are sixteen VMs, and they have been evenly distributed between the two nodes:

```
host:~ # numastat -p qemu-system

Per-node process memory usage (in MBs)
PID                             Node 0          Node 1            Total
----------------------  --------------- --------------- ---------------
200907 (qemu-system-x86)       43635.53           55.60         43691.13
201027 (qemu-system-x86)       43691.16           55.60         43746.76
201145 (qemu-system-x86)       43712.86           55.60         43768.46
201263 (qemu-system-x86)       43712.64           55.60         43768.24
201380 (qemu-system-x86)       43736.95           55.60         43792.55
201501 (qemu-system-x86)       43754.32           55.60         43809.93
201620 (qemu-system-x86)       43690.27           55.60         43745.87
201737 (qemu-system-x86)       43776.14           55.60         43831.75
201856 (qemu-system-x86)           0.00        43800.09         43800.09
201973 (qemu-system-x86)           0.00        43733.63         43733.63
202090 (qemu-system-x86)           0.00        43765.54         43765.54
202205 (qemu-system-x86)           0.00        43798.21         43798.21
202325 (qemu-system-x86)           0.00        43752.20         43752.20
202450 (qemu-system-x86)           0.00        43759.35         43759.35
202570 (qemu-system-x86)           0.00        43775.38         43775.38
202692 (qemu-system-x86)           0.00        43800.96         43800.96
----------------------  --------------- --------------- ---------------
Total                         349709.87       350630.17        700340.04
```

Placement of vCPUs happens via the `<cputune>` element, as in the example below:

```
<vcpu placement='static'>512</vcpu>
```

```
<cputune>
    <vcpupin vcpu="0" cpuset="0"/>
    <vcpupin vcpu="1" cpuset="256"/>
    <vcpupin vcpu="2" cpuset="1"/>
    <vcpupin vcpu="3" cpuset="257"/>

    ...
    <vcpupin vcpu="256" cpuset="128"/>
    <vcpupin vcpu="257" cpuset="384"/>
    <vcpupin vcpu="258" cpuset="129"/>
    <vcpupin vcpu="259" cpuset="385"/>

    ...
</cputune>
```

The value `512` means that the VM has 512 vCPUs. `static` guarantees that each vCPU will stay on the pCPU(s) on which it is "pinned" to. The various `<vcpupin>` elements are where the mapping between vCPUs and pCPUs is established (`vcpu` being the vCPU ID and `cpuset` being either one or a list of pCPU IDs).

To be able to create VMs with more than 255 vCPUs, the following element should be added in the `<device>` section:

```
<features>
  ...
  <ioapic driver="qemu"/>
</features>
...
<devices>
  ...
  <iommu model='intel'>
    <driver intremap='on'/>
  </iommu>
  ...
</devices>
```

When pinning vCPUs to pCPUs, adjacent vCPU IDs (like vCPU ID 0 and vCPU ID 1) must be assigned to host SMT siblings (like, for example, pCPU 4 and pCPU 260, on our test server):

```
host:~ # cat /sys/devices/system/cpu/cpu4/topology/thread_siblings_list
4,260
```

In fact, QEMU uses a static SMT sibling CPU ID assignment. This is how we guarantee that virtual SMT siblings will always execute on actual physical SMT siblings:

```
vm1:~ # cat /sys/devices/system/cpu/cpu8/topology/thread_siblings_list
8,9
```

The following sections give more specific and precise details about placement of vCPUs and memory for VM of varying sizes, on the reference system for this guide (see *Figure 1, "AMD EPYC 9005 Series Processors based on Zen5 cores Topology"*. Note that in SUSE Linux Enterprise 15 SP6, as an enhancement, as compared to previous OS versions, we can actually create VMs as large as our test host is, that means a VM with 512 virtual CPUs.

### 18.1.1   Placement of a single large VM

An interesting use case is when "only one" VM is used. Reasons for doing something like that include security/isolation, flexibility, high availability, and others. In this cases, typically, the VM is almost as large as the host itself.

Let us, therefore, consider a VM with 512 vCPUs and 672 GB of RAM. Such VM spans multiple host NUMA nodes and therefore it is recommended to create a virtual NUMA topology for it.

We should create a virtual topology with 2 virtual NUMA nodes (that is, as many as there are physical NUMA nodes), and split the VM's memory equally among them. The 672 vCPUs are assigned to the 2 nodes, 336 on each. This way, if a suitable virtual topology is also provided to the VM, each of the VM's vCPUs will access its own memory directly, and use Infinity Fabric inter-socket links only to reach foreign memory, as it happens on the host. Workloads running inside the VM can be tuned exactly like they were running on a bare metal 5th Generation AMD EPYC Processor server.

The following example `numactl` output comes from a VM configured as explained:

```
vm1:~ # numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
 58
 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
 88
 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157
 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201
 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245
 246 247 248 249 250 251 252 253 254 255
node 0 size: 338489 MB
node 0 free: 337385 MB
node 1 cpus: 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274
```

```
275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296
297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318
319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362
363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384
385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406
407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428
429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450
451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472
473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494
495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511
node 1 size: 338432 MB
node 1 free: 337252 MB
node distances:
node   0   1
  0:  10  32
  1:  32  10
```

This is analogous to the host topology, also from the point of view of NUMA node distances (as the `libvirt` version present in SUSE Linux Enterprise Server 15 SP6 allows us to define the virtual nodes distance table). The only differences are the APIC IDs of the CPUs and the amount of memory.

See *Section 23, "Appendix A"* for a complete VM configuration file.

As said already, full cores must always be used. If possible always fully use CCXes/dies too. Since each die has 16 CPUs, that means that a VM with 512 vCPUs will use all the 16 CCXes, on each of the 2 nodes (as 16 x 16 x 2 is indeed 512). So, for example, vCPUs 0 to 15 can be assigned to Cores L#0 to L#7 (and hence to CPUs P#0 to P#7 and P#256 to P#263), on node P#0; vCPUs 16 to 31 to Cores L#$8 to L#15, and so on (and the same on node P#1). In fact, this is what we call coherent 1-to-1 mapping between virtual and physical topologies.

## 18.1.2   Placement of two large VMs

If wanting to run two VMs, they both can have 256 vCPUs and 336 GB memory each. Therefore, each one can fit in one of the host's NUMA nodes. This also means that there is no need to define virtual NUMA nodes in their configuration.

Placing each VM on one node means that workloads running inside each of them will never need to use inter-socket interconnect.

In this example scenario, `numactl` output of both VMs looks as follows:

```
vm1:~ # numactl --hardware
available: 1 nodes (0)
```

```
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
 58
 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
 88
 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157
 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201
 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245
 246 247 248 249 250 251 252 253 254 255
node 0 size: 338246 MB
node 0 free: 337317 MB
node distances:
node   0
  0:  10
```

### 18.1.3   Placement of four to thirty-two medium-size VMs

Following the same principles, we can "split" each node in 2, and have 4 VMs with 128 vCPUs and up to 168 GB of RAM each. It is also feasible to have 8 VMs with 64 vCPUs and up to 84 GB of RAM each, 16 VMs with 32 vCPUs and 42 GB RAM, or even 32 VMs, all with 16 vCPUs and 21 GB RAM.

In the first case, each VM will span 8 host dies. In the second and in the third cases, 4 and 2 dies. And in the fourth one, each VM will be pinned to 1 die.

In all these situations, VMs do not need to (and, therefore, should not) be placed across the host NUMA nodes boundary, and hence they do not need to be NUMA-aware. They still benefit from having a virtual topology, in terms of cores, threads and dies which matches the resource they are using (discussed in more details about in the following section).

### 18.1.4   Placement of many small VMs

If small VMs are the target, VMs with either 8, 4 or 2 vCPUs each can be created and efficiently allocated on AMD EPYC 9005 Series Processors servers.

VMs with 8 vCPUs "occupies" half a die, and we can have 64 of them. VMs with either 4 or 2 vCPUs will be given 2 and 1 host cores and we can have 128 and 256 of them, respectively. And this is all possible without the need for any of the VMs to span more than one host NUMA node.

Of course, in all these cases, VMs are sharing dies, which means VMs will interfere with each other via the L3 caches. This may or may not be a problem, but there is no way around it, as soon as more VMs than the number of available dies are necessary. The performance impact of such sharing should be tolerable, in most cases, for these configurations, but this needs to be assessed with tests and benchmarks. If that is not the case, then the recommendation is to not go above 32 VMs. More clever (but also more complex) tuning strategies can be designed, but they would require further a-priori knowledge of the workload and/or the load profile of the various VMs, which may not be always available (or can change during the lifecycle of the VMs).

### Note: Hundreds of VMs with limited RAM size and disk space

When the number of VM increases so much, the limiting factor will become the memory or the disk. With ~700 GB of RAM available for VMs on our server, we can give only 2 GB of RAM to each of the 256 VMs of the last example above. Of course, this limitation is only an issue of the system used as reference for this guide. The AMD EPIC 9005 Series Processor architecture itself can accommodate much more RAM.

In cases when it is enough or desirable to have VMs with only 1 vCPUs, it is recommended to still not go beyond 256, and assign no less than one full core to each VM. In fact, it should be avoided to have vCPUs from different VMs running on two siblings threads of the same core. Such a solution, although functional, is not ideal for all the cases where good and consistent performance within VMs is a goal (as well as for security concerns, but that is out of the scope of this guide). What can be done for each VM and for each core is to use one of the two threads for the actual vCPU, and the other for its IO thread(s), on the host. This solution allows to take advantage of the full 5th Generation AMD EPYC processing power. However, we advise to check and verify whether this is a good setup and if it brings performance benefits for the specific workloads of interest, before committing to it.

## 18.2   Emulator IO threads & disaggregation

IO for the VMs is carried out by device emulators running on the host, or by the so-called *back-ends* of paravirtualized drivers (also running on the host). Both the IO threads of the emulators and the back-ends are seen as (either user or kernel) threads by the host OS. As such, they can be configured to run on specific subsets of the CPUs assigned to the host OS (Dom0's virtual CPUs, in the case of Xen). And their placement on such CPUs may have an impact on performance of VMs, especially on IO bound workloads.

For example, a 1 vCPU VM can have its vCPU bound to the one SMT thread of a core, while the other thread can be assigned to the host, and the IO thread(s) of the VM pinned to it. The idea behind this is that a common execution pattern for the vCPU is to be idle when the IO threads are busy. Hence this setup potentially maximizes the exploitation of hardware resources.

On Xen, there is also the possibility of setting up driver domains. These are special VMs which act as back-ends of a particular IO device, for one or more VMs. In case they are used, make sure that they run "close" enough to both the hardware they are providing their abstraction for, and the VMs that they are servicing.

## 18.3   Oversubscription of host resources

Oversubscription happens when the demand for some resource is higher than the resource is physically available. In virtualization, this can happen for CPU and memory.

> ### 🔖 Note: Not covering oversubscribed scenarios
>
> This guide does not cover in great details oversubscribed scenarios. However, given the large number of CPUs available on an AMD EPYC 9005 Series Processor system and the huge amount of memory the architecture supports, this is not considered a limitation.

### CPU Oversubscription

CPU oversubscription happens when the total cumulative number of vCPUs from all VMs becomes higher than 512. Such a situation inevitably introduces latencies, resulting in lower performance compared to when host resources are sufficient. It is, however, impossible to tell a priori by what extent this happens, at least not without a detailed knowledge of the actual workload.

If we knew in advance that the load on each vCPU will always stay below 50%, we could assume that even oversubscribing the host by a factor of 2 (like with ~1024 vCPUs in total, in our case!) will work well. On the other hand, if we knew that the load that each vCPU tries to impose on the system is always 100%, creating even 1 vCPUs more than the host has pCPUs can be considered a misconfiguration.

When oversubscription is necessary, exclusive 1-to-1 vCPU to pCPU assignment may not the best approach and we need to trust the hypervisor scheduler to handle the situation well. What can be done, though, is providing such scheduler at least with some "suggestions", if we have

enough knowledge about the workload. For example, we can try to help the scheduler avoiding that all the VMs running CPU intensive workloads end on the same nodes, or avoiding cross-node VM migrations happening too frequently.

This grouping of VMs on (a set of) nodes can be done with some less strict forms of vCPU pinning (often called "affinity"), or with other mechanisms. For example, `cgroups` can be leveraged on a KVM host, or `cpupools` on a Xen host. Xen also contains a feature, called *soft vCPU affinity*, which can be used together with "traditional" vCPU affinity (also called *hard vCPU affinity*) as a finer grained and more powerful way of controlling resource allocation in oversubscribed scenario.

## Memory Oversubscription

Memory oversubscription happens when the total cumulative amount of memory used by all VMs is more than the RAM available on the host. In this case, some of such memory is kept outside of the RAM (*swapped out*) when the VMs using it are not running. It is put back in RAM (*swapped in*) when these VMs run again. This also has (potentially severe) performance implications, and is not analyzed in details in here.

On KVM, swapping is handled by Linux virtual memory management and paging code and mechanism, exactly as it happens for host/bare metal processes. On Xen, this is not supported unless special technologies (`xenpaging` or transcendent memory) are employed.

Then there are page sharing and memory ballooning.

Page sharing relies on the principle that, when VMs use memory pages which are identical, only one copy of them needs to be kept in physical RAM, at least for most of the time. This is supported on KVM, via a mechanism called Kernel Samepage Merging (KSM). On Xen, this again require special tools and actions.

Ballooning relies on the idea that VMs do not always use all the memory they are configured to be able to use. This is supported in both Xen and KVM. However, using it is not ideal for performance, thus it is not analyzed in any further details in this document.

## Oversubscription with a Single VM

This is always considered a misconfiguration. In fact:

- A single VM should never have more vCPUs than the host has pCPUs.

- A single VM should never have more memory than the host has physical RAM.

## 18.4 Enlightenment of VMs

The term "enlightenment" refers to letting the guest OS know as many details as possible about the virtual topology of the VM. This is only useful and brings actual performance improvements *only* if such topology is properly and effectively mapped on host resources, and if such mapping is stable and does not change during the life of the VM.

### 18.4.1 Virtual CPUs model and topology

To ensure the VM has a vCPU topology, use the following:

```
<cpu mode="host-model" check="none">
<topology sockets="2" dies="16" cores="8" threads="2"/>
<numa>
  <cell id="0" cpus="0-255" memory="352321536" unit="KiB">
    <distances>
      <sibling id="0" value="10"/>
      <sibling id="1" value="32"/>
    </distances>
  </cell>
  <cell id="1" cpus="256-511" memory="352321536" unit="KiB">
    <distances>
      <sibling id="0" value="32"/>
      <sibling id="1" value="10"/>
    </distances>
  </cell>
</numa>
</cpu>
```

The `<topology>` element specifies the CPU characteristics. In this case, we are creating vCPUs which will be seen by the guest OS as being arranged in 2 sockets, each of which has 16 dies, each of which has 8 cores with 2 threads (that is 16 CPUs). And this is how we match, for the one big VM, the topology of the host.

Each `<cell>` element defines one virtual NUMA node, specifying how much memory and which vCPUs are part of it. The `<distances>` elements allow us to define a custom virtual NUMA distance table. Values should be - as it is on actual hardware - an indication of the latency of accessing the memory of every node from any other. And again we are defining a virtual distance table which exactly matches the host one, for the one big VM.

CPU model is how the VM is told what type of vCPUs should be "emulated", for example, what features and special instructions and extensions will be available. To achieve best performance, using `model='host-passthrough'` is often the right choice. However, at least on SUSE Linux

Enterprise Server 15 SP6, this produces wrong results. In fact, such setting prevents the topology - despite it being described correctly in the `<topology>` element - from being interpreted correctly, inside the VM. This is visible in the output of the command **lscpu**:

```
vm1:~ # lscpu
...
CPU(s):                 512
  On-line CPU(s) list:  0-511
Vendor ID:              AuthenticAMD
  Model name:           AMD EPYC 9755 128-Core Processor
    CPU family:         26
    Model:              2
    Thread(s) per core: 1
    Core(s) per socket: 512
    Socket(s):          1
...
Caches (sum of all):
  L1d:                  32 MiB (512 instances)
  L1i:                  32 MiB (512 instances)
  L2:                   256 MiB (512 instances)
  L3:                   8 GiB (512 instances)
...
```

Both the CPU topology and the cache layout, as seen from inside of the VM, are completely wrong, and can affect performance of applications running inside the VM. In fact, not only the sizes of the various cache levels are wrong. It is also the information about which CPUs share what caches that is completely misconfigured, as shown also here:

```
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index0/shared_cpu_list
0
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index1/shared_cpu_list
0
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index2/shared_cpu_list
0
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index3/shared_cpu_list
0-255
```

The VM thinks that L1d, L1i and L2 are private of each vCPU, while that is not the case, as they are shared by the two threads of each core. Even worse, it thinks that L3 is shared across all the vCPUs of a socket (and that there is only one socket!), which is not true, as it is shared only among 8 cores (that is 16 vCPUs).

For this reason, the recommendation is to use `EPYC-Genoa` as the CPU model (as an appropriate model for 5th Generation AMD EPYC Processors is not yet available) either manually or by using `model='host-model'`. This enables a virtual topology inside the VM what is more consistent

(although not perfectly identical) with the one of the host, and benchmarks show a beneficial effect on performance coming from that. The output of **lscpu** with `EPYC-Genoa` as a model is shown below:

```
vm1:~ # lscpu
...
CPU(s):                  512
  On-line CPU(s) list:   0-511
Vendor ID:               AuthenticAMD
  Model name:            AMD EPYC-Genoa Processor
    CPU family:          25
    Model:               17
    Thread(s) per core:  2
    Core(s) per socket:  128
    Socket(s):           2
...
Caches (sum of all):
  L1d:                   8 MiB (256 instances)
  L1i:                   8 MiB (256 instances)
  L2:                    256 MiB (256 instances)
  L3:                    1 GiB (32 instances)
...
```

Of course, now it is the `Model name:` that is not completely correct. But both the CPU and caches layout is much more in line with what we wanted to achieve by tuning the VM configuration.

In fact, even if the cache sizes that are slightly off, with regard to the host value. And this is because they are the values taken from previous generation AMD EPYC Processors architecture characteristics. But the cache topology and cache level sharing among vCPUs are now correct (as shown below). And this is far more important for achieving good performance inside the VM.

```
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index0/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index1/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index2/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index3/shared_cpu_list
0-15
```

Note that, to achieve the outcome shown above, it is very important to use the following CPU topology description string, in the VM configuration:

```
<topology sockets="2" dies="16" cores="8" threads="2"/>
```

In particular, do not omit the `dies="16"` element, as that is what enables the VM to see its vCPUs grouped in dies (the CCXes of the host). In fact, using something like this `<topology sockets="2" cores="128" threads="2"/>` would result in a correct CPU topology, but the representation of the cache layout would still be inaccurate, such as:

```
vm1:~ # lscpu
...
CPU(s):                 512
  On-line CPU(s) list:  0-511
Vendor ID:              AuthenticAMD
  Model name:           AMD EPYC-Genoa Processor
    CPU family:         25
    Model:              17
    Thread(s) per core: 2
    Core(s) per socket: 128
    Socket(s):          2
...
Caches (sum of all):
  L1d:                  8 MiB (256 instances)
  L1i:                  8 MiB (256 instances)
  L2:                   256 MiB (256 instances)
  L3:                   64 MiB (2 instances)
...
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index0/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index1/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index2/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index3/shared_cpu_list
0-255
...
```

As a further example, in case we have two VMs, one for each NUMA node and with 256 vCPUs each, both are also configured to have 16 dies (but, of course, just 1 socket), like this:

```
<topology sockets="1" dies="16" cores="8" threads="2"/>
```

And the topology, from inside each VM, will look as follows:

```
vm1:~ # lscpu
...
CPU(s):                 256
  On-line CPU(s) list:  0-255
Vendor ID:              AuthenticAMD
  Model name:           AMD EPYC-Genoa Processor
```

```
    CPU family:          25
    Model:               17
    Thread(s) per core:  2
    Core(s) per socket:  128
    Socket(s):           1
...
Caches (sum of all):
  L1d:                   4 MiB (128 instances)
  L1i:                   4 MiB (128 instances)
  L2:                    128 MiB (128 instances)
  L3:                    512 MiB (16 instances)
...

vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index0/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index1/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index2/shared_cpu_list
0-1
vm1:~ # cat /sys/devices/system/cpu/cpu0/cache/index3/shared_cpu_list
0-15
```

Finally, on SUSE Linux Enterprise Server 15 SP6 it is possible to select the `haltpoll` CPU-Idle governor, inside the VM, even if we are using `model='host-model'` (although, it is only partially effective, as `model='host-passthourgh'` would be necessary in order for the VM to be able to exploit this feature at its full potential).

This is an optimization meant at reducing the number of context switches between the VM and the host (also known as VMExits), when doing static resource partitioning of the host itself. For that reason, it is something that we recommend using, although the actual benefit provides highly depends on the workload running inside of the VM.

For enabling it, just load the kernel module (inside of the VM, of course):

```
vm1:~ # modprobe cpuidle-haltpoll
```

### 18.4.2   Memory backing

Memory wise, the VMs must be told to use the Huge Pages that were reserved for them. To effectively use 1 GB huge pages, the amount of memory each VM is given must be a multiple of 1 GB. Also, KSM should be disabled and we also must ensure that VMs' memory is never going to be swapped out. This is all achieved as follows:

```
<memory unit='KiB'>704643072</memory>
```

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
  </hugepages>
  <nosharepages/>
</memoryBacking>
```

To verify that the appropriate type of memory is being used by the VMs, one can check the content of `/proc/meminfo`, with the VMs running, and observe that all the pre-allocated Huge Pages are actually occupied.

```
host:~ # cat /proc/meminfo | grep Huge
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:     672
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB
Hugetlb:       704643072 kB
```

### ✎ Note: Huge Pages, shared pages and locked pages on Xen

On Xen, for HVM guests, huge pages are used by default and there is neither any page sharing nor swapping (at least not in SUSE Linux Enterprise Server 15 SP6). Therefore, the entire `<memoryBacking>` element is technically not necessary.

## 18.4.3   Ballooning

To get the full benefit of Huge Pages, memory ballooning must also be disabled. In fact, if the ballooning driver is not Huge Pages-aware, using ballooning would end up splitting the pages in fragments, neutering their (positive) performance impact. Disabling memory ballooning is done as follows:

```
<memory unit='KiB'>704643072</memory>
<currentMemory unit='KiB'>704643072</currentMemory>
```

That is, by specifying in the `<currentMemory>` the same value already used in the `<memory>` element (and never changing the memory size of the VM at runtime).

### 18.4.4    (Transparent) Huge Pages

If huge pages are used for allocating the VMs' memory on the host, they can also be used inside the VMs, either explicitly, or via THP. Whether that helps performance is workload dependent. The analysis and the considerations made in the first part of the document about using (T)HP on bare metal, can also be applied here.

### 18.4.5    Automatic NUMA balancing

Similarly to THP, if the VM is NUMA-aware, NUMAB can be used inside of it to boost the performance of most NUMA-unaware workloads running inside the VM itself.

### 18.4.6    Services and daemons

`irqbalance` can be a source of latency inside of the VM, because of the way it uses the `/proc/interrupts` interface. For workloads that are particularly sensitive to latency, consider disabling it within the VMs (of course by taking the appropriate alternative measures, like binding IRQs, if necessary).

## 18.5    Secure Encrypted Virtualization with Encrypted State

The following documents:

- Libvirt Documentation: Launch security with AMD SEV (https://libvirt.org/kbase/launch_security_sev.html) ↗

- SUSE Linux Enterprise Server 15 SP4: AMD Secure Encrypted Virtualization (AMD-SEV) Guide (https://documentation.suse.com/sles/15-SP4/html/SLES-amd-sev/article-amd-sev.html) ↗

Explain how to configure a VM to use AMD SEV-ES (on an AMD SEV-ES capable and properly configured host) is not in the scope of this document. Refer to the linked materials for the details.

It is possible to check if the host is properly configured to run SEV-ES VMs with the following command:

```
[    0.000000] SEV-SNP: Memory for the RMP table has not been reserved by BIOS
[   33.954913] ccp 0000:55:00.5: SEV API:1.55 build:40
[   33.983047] ccp 0000:55:00.5: SEV API:1.55 build:40
```

```
[   41.100388] kvm_amd: SEV enabled (ASIDs 36 - 1006)
[   41.106817] kvm_amd: SEV-ES enabled (ASIDs 1 - 35)
```

Once inside of the VM, this is how one can check whether it is only the memory that is being encrypted (with SEV):

```
dmesg | grep SEV
[    0.069436] AMD Memory Encryption Features active: SEV
```

Or if we are also fully encrypting the VM state (with SEV-ES):

```
dmesg | grep SEV
[    0.069436] AMD Memory Encryption Features active: SEV SEV-ES
```

### SEV and The IOMMU Device

Note that it is not possible to provide a virtual IOMMU device to an encrypted VM. This means that, for having a functional SEV or SEV-ES VM, we need to remove the `iommu` element, in the `<device>` section of the VM configuration. And since that element is necessary for having more than 255 vCPUs in the VM, all the experiments conducted with SEV enabled have been done in VMs with at most 128 vCPUs.

# 19   Test VM workload: STREAM

The effectiveness of the proposed tuning is demonstrated using the STREAM benchmark.

## 19.1   Test scenario: One large VM

*Figure 4, "STREAM Bandwidth - Bare metal compared with one VM"* shows the bandwidth achieved by STREAM when the benchmark is built in single thread mode ('single') or parallelized with OpenMP ('openmp'). For both cases, we compare the results achieved on the host ('BM') and inside one VM ('VM').
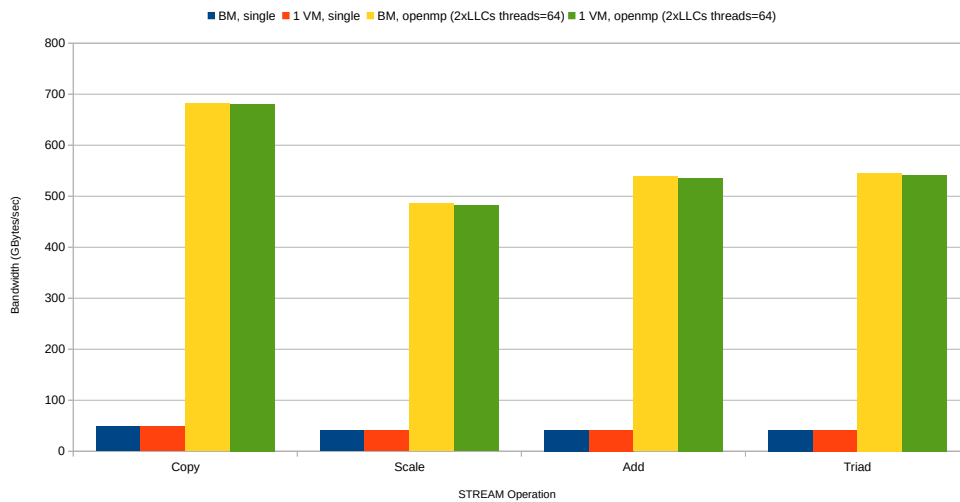
Both the single thread and the parallel results are identical (within 1% difference!) between bare metal (blue and yellow rectangles) and inside of the VM (orange and green rectangles), for all the operations (`Copy`, `Scale`, `Add` and `Triad`) of the benchmark.

This clearly shows how proper tuning allows a single VM running on an AMD EPYC 7005 Series Processor server to achieve a memory bandwidth that matches the one that we can reach directly on the host.

## Note

Inside of the VM, the STREAM benchmark was configured almost identically to what has been shown already in *Section 13.2, "Test workload: STREAM"*.
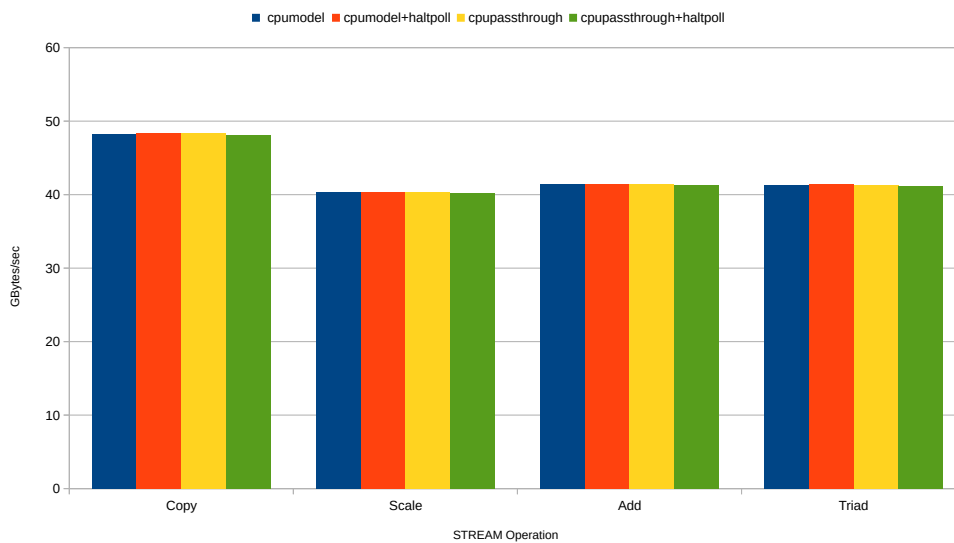
*Figure 5, "STREAM Bandwidth - Single thread in one VM with different CPU models"* and *Figure 6, "STREAM Bandwidth - OpenMP in one VM with different CPU models"* show the effect of using different CPU models for the virtual machine. We see that, as long as a single thread is used, the CPU model chosen for the VM is completely irrelevant.

On the other hand, the OpenMP run clearly shows some problems when the `host-passthrough` CPU model is selected. In fact, since using that model builds a VM with only 2 LLCs (and also because of the other problems with the cache topology), running STREAM with twice as many threads as there are LLCs in the system results in the benchmark spawning only 4 of them (yellow and green rectangles). And this, of course, dramatically reduces the performance. We can also see that, if we instead manually set the number of threads to the value that we know to be the

best for this VM (that is 64, dark red and cyan rectangles) performance are restored to how good we know things can be from *Figure 4, "STREAM Bandwidth - Bare metal compared with one VM"* (and from the `cpumodel` results, see the blue and orange rectangles).

It is also interesting to note that with `cpumodel+haltpoll` (orange rectangles) we reach the same performance of the configuration that should theoretically be the absolute best one (`cpupassthrough+haltpoll (64 threads)`, cyan rectangle). This means that it is fine to stick with the former, and it is not worthwhile setting `cpupassthrough` and coping manually with the misconfigured cache hierarchy inside of the VM (as that could potentially be much more complicated it has been here for the STREAM benchmark or, sometimes, even impossible).



**FIGURE 5: STREAM BANDWIDTH - SINGLE THREAD IN ONE VM WITH DIFFERENT CPU MODELS**

Legend:
- cpumodel (2xLLCs threads=64)
- cpumodel+haltpoll (2xLLCs threads=64)
- cpupassthough (2xLLCs=4 threads)
- cpupassthough+haltpoll (2xLLCs=4 threads)
- cpupassthrough (64 threads)
- cpupassthrough+haltpoll (64 threads)
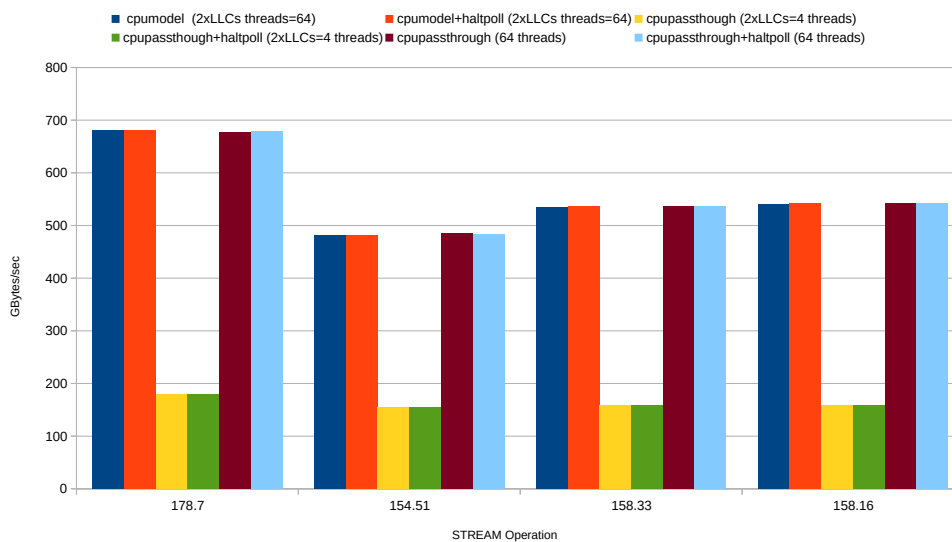
**FIGURE 6: STREAM BANDWIDTH - OPENMP IN ONE VM WITH DIFFERENT CPU MODELS**

This situation also confirms the need to always run benchmarks for assessing the performance of the relevant workloads, on a given platform. In fact, because of specific characteristics of some components of the virtualization stack available in SUSE Linux Enterprise Server 15 SP4, the `EPYC-Milan` (or, equivalently, the `host-model`) CPU model might be preferable to what it would have appeared to be the most obvious choice (`host-passthrough`).

Finally, about the CPUIdle `haltpoll` governor, it is recommended to enable and make use of it, although for STREAM (and we can guess also for other, similar, memory intensive benchmarks) it introduces no significant performance difference.

## 19.2  Test scenario: Multiple VMs

*Figure 7, "STREAM Bandwidth - Single, average of the bandwidth achieved among all VMs of each group"* and *Figure 8, "STREAM Bandwidth - Single, sum of the bandwidth achieved within all VMs of each group"* show what happens when 1, 2, 4, 8, 16 and 32 VMs are used. The former reports the average bandwidth achieved in each experiments, considering all the VMs involved. For example, the yellow blue bar (4 VMS) is relative to an experiment where 4 VMs were concurrently running the STREAM benchmark (in single thread mode) and represents the average of the 4 different memory bandwidth values, one for each of such VMs. The latter shows something similar, but the bars represents the cumulative STREAM bandwidth achieved in each experiments, considering all the VMs involved. For example, the cyan bar (32 VMs) is relative to an experiment where 32 VMs were concurrently running the STREAM benchmark (in single thread mode) and represents the sum of the 24 different memory bandwidth values, one of each of such VMs.
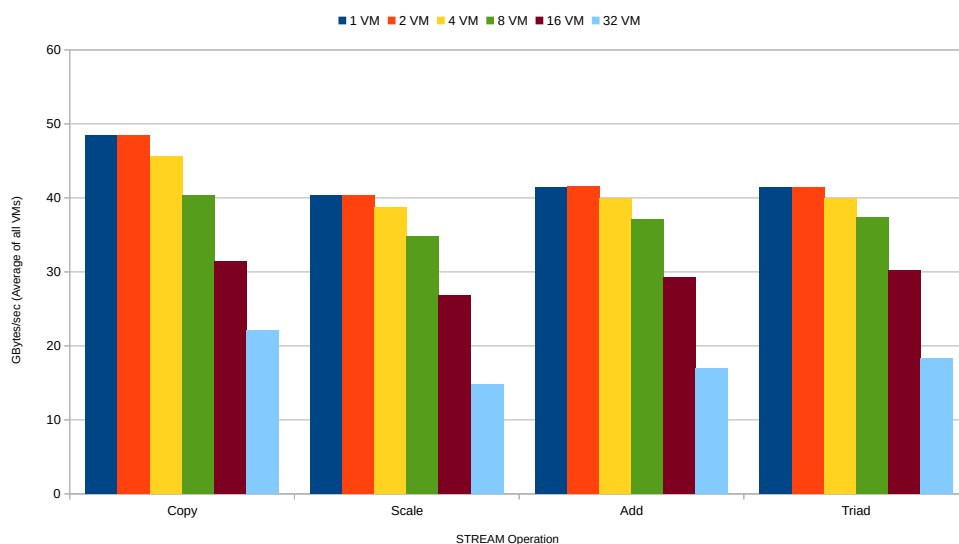
FIGURE 7: STREAM BANDWIDTH - SINGLE, AVERAGE OF THE BANDWIDTH ACHIEVED AMONG ALL VMS OF EACH GROUP
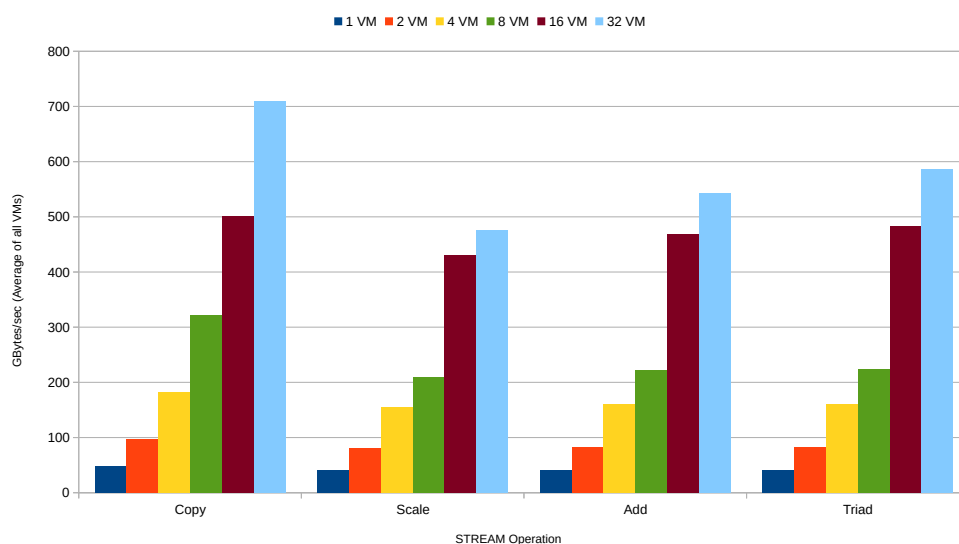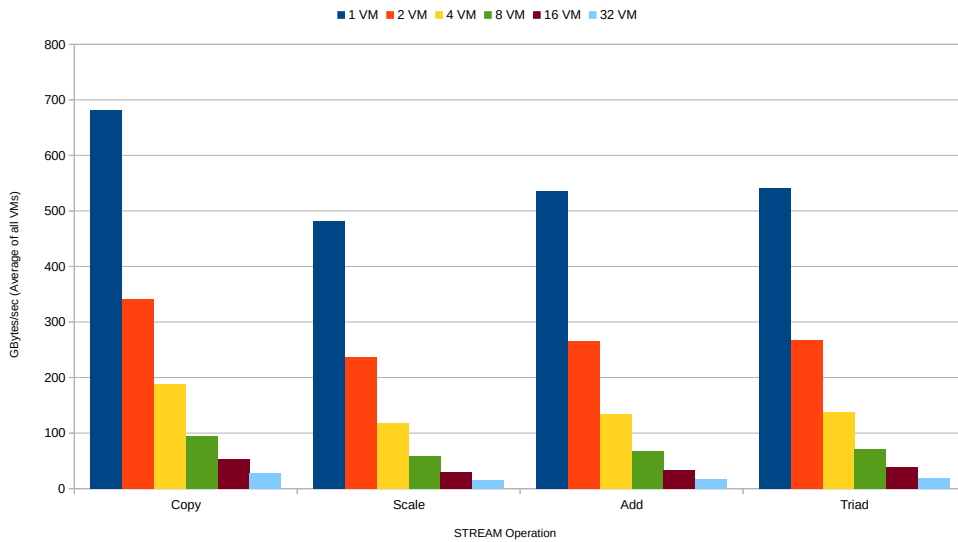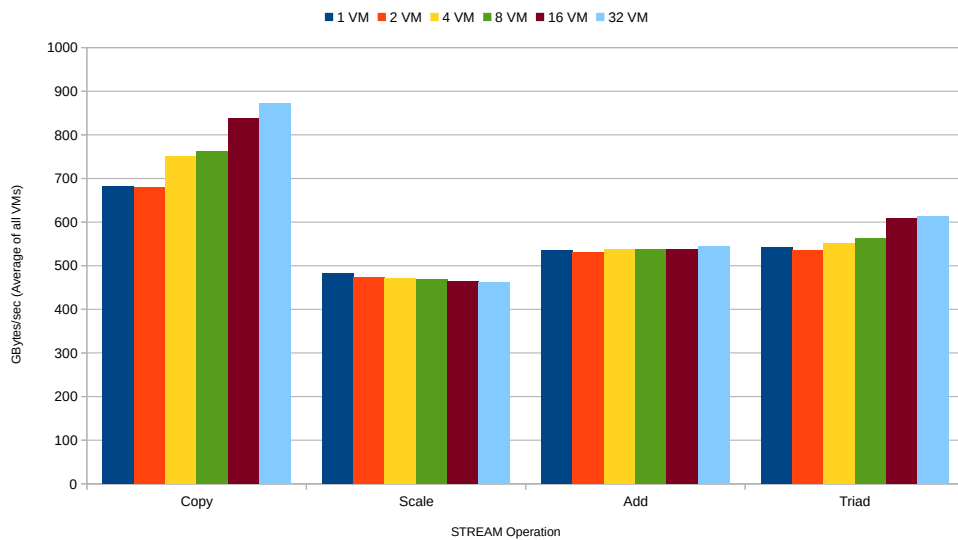


FIGURE 8: STREAM BANDWIDTH - SINGLE, SUM OF THE BANDWIDTH ACHIEVED WITHIN ALL VMS OF EACH GROUP

We see in *Figure 7, "STREAM Bandwidth - Single, average of the bandwidth achieved among all VMs of each group"* how the single STREAM bandwidth suffers a bit of a decline as more VMs are packed on the NUMA nodes and, hence, compete for the bandwidth of the memory controllers. However, *Figure 8, "STREAM Bandwidth - Single, sum of the bandwidth achieved within all VMs of each group"* reminds us how the total bandwidth achieved, if we consider all the VMs involved in each

Optimizing Linux for AMD EPYC™ 9005 Series Processors with SUSE Linux Enterprise 15 SP6

experiments, actually goes up (and it scales roughly linearly, at least for some of the STREAM operations), until it reaches the same level that we know (for example, from *Figure 4, "STREAM Bandwidth - Bare metal compared with one VM"*) it can touch.

*Figure 9, "STREAM Bandwidth - OpenMP, average of the bandwidth achieved among all VMs of each group"* and *Figure 10, "STREAM Bandwidth - OpenMP, sum of the bandwidth achieved within all VMs of each group"* show the same, but when the parallel (via OpenMP) version of STREAM is run, inside of each VM of each experiment.



**FIGURE 9: STREAM BANDWIDTH - OPENMP, AVERAGE OF THE BANDWIDTH ACHIEVED AMONG ALL VMS OF EACH GROUP**



**FIGURE 10: STREAM BANDWIDTH - OPENMP, SUM OF THE BANDWIDTH ACHIEVED WITHIN ALL VMS OF EACH GROUP**

Optimizing Linux for AMD EPYC™ 9005 Series Processors with SUSE Linux Enterprise 15 SP6

In *Figure 9, "STREAM Bandwidth - OpenMP, average of the bandwidth achieved among all VMs of each group"* we see that performance scales down in a close to linear fashion with the increase of the number of VMs involved (although, not perfectly linearly for all the type of STREAM operations). At the same time, *Figure 10, "STREAM Bandwidth - OpenMP, sum of the bandwidth achieved within all VMs of each group"* shows how the total cumulative bandwidth of each experiments stays pretty much always at its top levels, as one would expect. Actually, it even seems that using many small VMs may make it possible to exploit even better (as compared to what happens when few large VMs are running) the large memory bandwidth provided by the memory controllers of the AMD EPYC 9005 Series Processors.

## 19.3    Test scenario: Secure Encrypted Virtualization

The goal for this test was evaluating the impact on the performance of encrypting the VMs with SEV and with SEV-ES, when running a memory intensive workload like STREAM.

Considering the case when 4 VMs are concurrently running on the host and are executing the STREAM benchmark, *Figure 11, "STREAM Bandwidth - Single, average bandwidth in 4 VMs when unencrypted, encrypted with SEV and with SEV-ES"* shows the average bandwidth of the single thread execution of the benchmark in both such VMs when memory is not encrypted (blue bars), when memory is encrypted with SEV (orange bar) and with both memory and state are encrypted with SEV-ES (yellow bar).
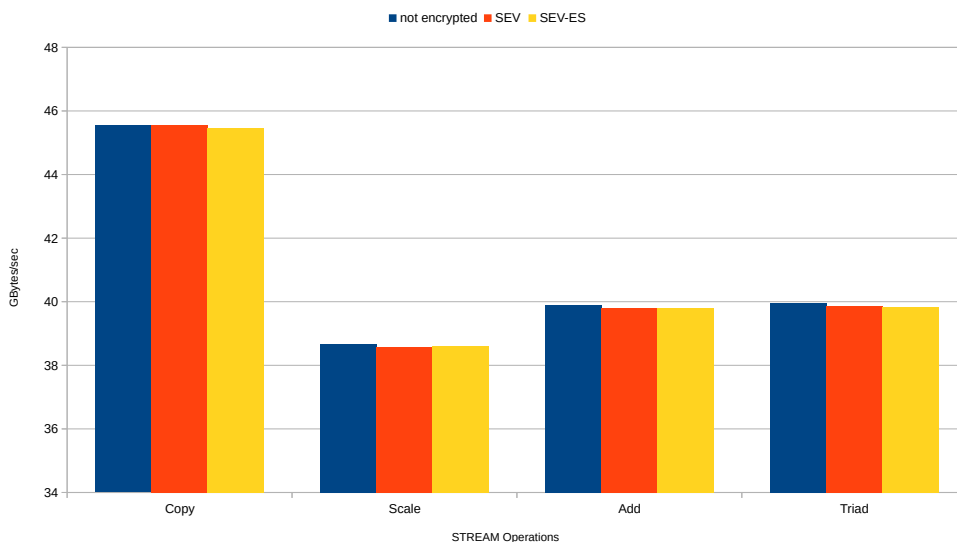


**FIGURE 11: STREAM BANDWIDTH - SINGLE, AVERAGE BANDWIDTH IN 4 VMS WHEN UNENCRYPTED, ENCRYPTED WITH SEV AND WITH SEV-ES**
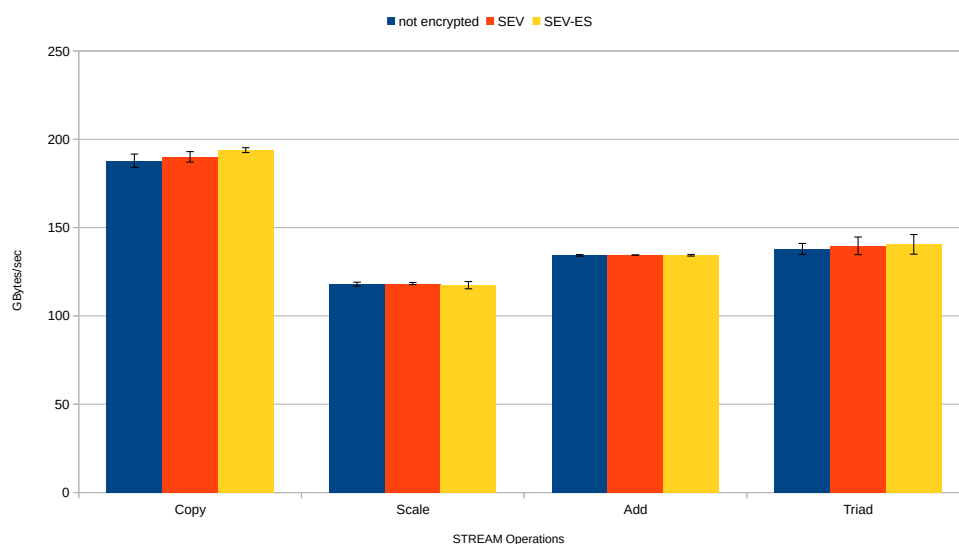
**FIGURE 12:** STREAM BANDWIDTH - OPENMP, AVERAGE BANDWIDTH IN 4 VMS WHEN UNENCRYPTED, ENCRYPTED WITH SEV AND WITH SEV-ES

It is quite evident how both SEV and SEV-ES have a close to nonexistent impact on the performance for this workload. Actually, it may even seem that, in the OpenMP case, encrypted VMs are faster, but we should note how all the results fall within the other ones' error bars.

# 20   Test VM workload: NPB

Trying to mach the experimental evaluation done on bare-metal, NPB base workloads have also been considered.

## 20.1   Test scenario: One large VM

*Figure 13, "NAS Completion Time - bare metal and in one VM"* shows the results of running the various NPB workloads on the host and in one VMs. In both cases, the tuning introduced above was applied (in the latter case, of course, inside of the VM itself).
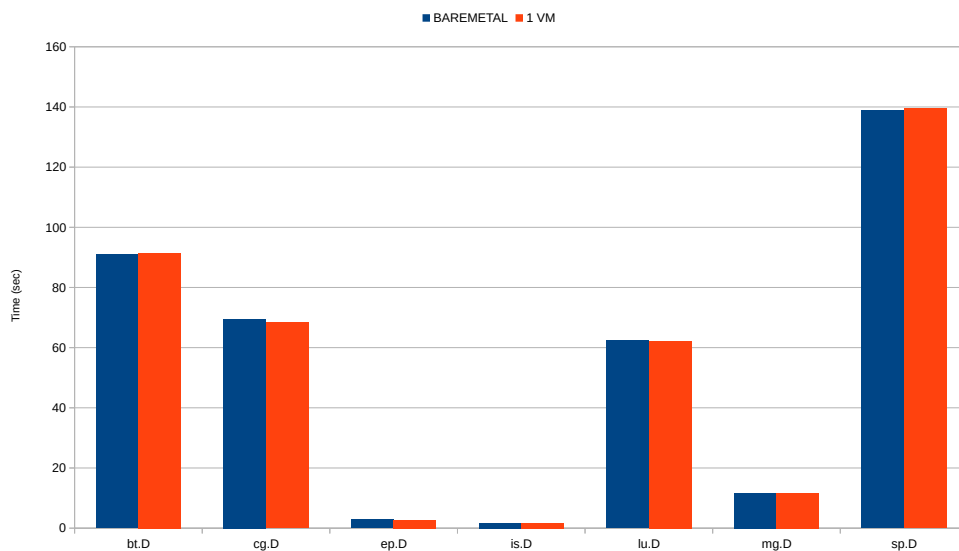
**FIGURE 13:** NAS COMPLETION TIME - BARE METAL AND IN ONE VM

Like for the STREAM case, a properly tuned VM can reach pretty much exactly the performance of the host.

*Figure 14, "STREAM Bandwidth - Single thread in one VM with different CPU models"* show again the impact on the performance of different CPU models.
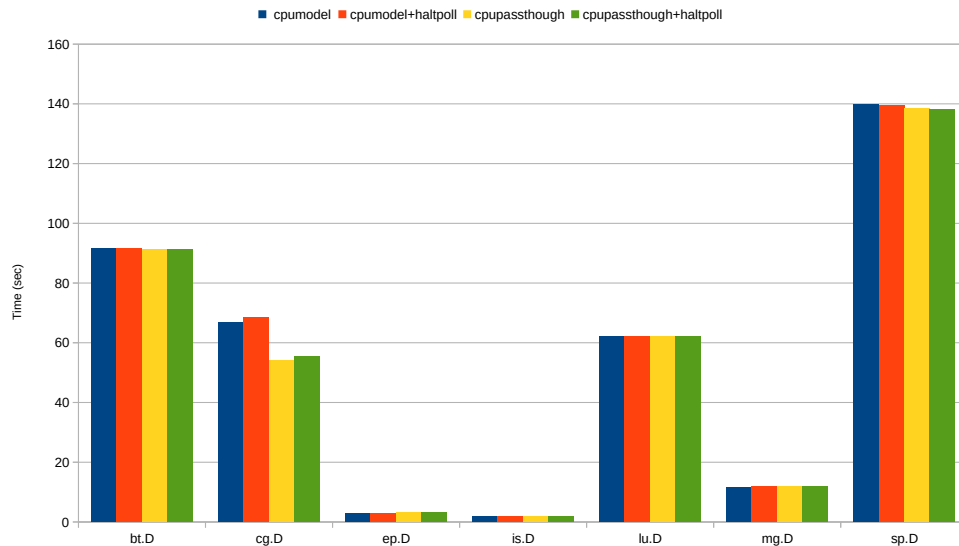


**FIGURE 14:** STREAM BANDWIDTH - SINGLE THREAD IN ONE VM WITH DIFFERENT CPU MODELS

Optimizing Linux for AMD EPYC™ 9005 Series Processors with SUSE Linux Enterprise 15 SP6

All the results are very similar and the only benchmark(s) where the `cpupassthrough` model seems to have an edge is `cg.D` (and, but only slightly, `sp.D`). Therefore, as already said for STREAM, considering that `cpumodel` runs show extremely good results, this guide recommends using it and avoiding the potential issues coming from having an inaccurate topology and a misleading cache hierarchy representation inside of VMs.

## 20.2   Test scenario: Two to twelve VMs

*Figure 15, "NAS Completion Time - bare metal and 1 to 16 VMs"* depicts the same as *Figure 13, "NAS Completion Time - bare metal and in one VM"*, but for a few more cases, in terms of number of concurrently running VMs.
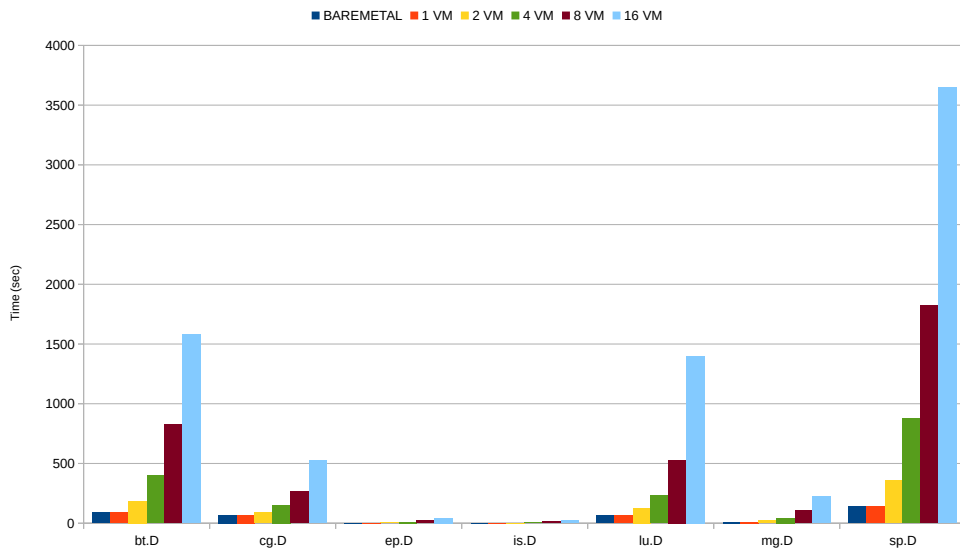


FIGURE 15: NAS COMPLETION TIME - BARE METAL AND 1 TO 16 VMS

We can see how the slowdown is linear (and sometimes even sub-linear) with the number of the VMs (the only exceptions being `lu.D` with 16 VMs). And that makes sense considering that when we increase the number of VMs, each one of them can also have fewer CPUs. This testifies the effectiveness of the suggested tuning measures, when it comes to guarantee a good level of isolation among multiple concurrently running VMs on the AMD EPYC 9005 Series Processors.

## 20.3 Test scenario: Two VMs with Secure Encrypted Virtualization

Similarly to what has been done with STREAM, we evaluate here the overhead introduced by SEV and SEV-ES, this time for CPU-bound workloads, and we do that by running the NPB benchmarks it inside two VMs concurrently that are all either unencrypted, encrypted with SEV or encrypted with SEV-ES (*Figure 16, "NAS Completion Time - Tuned configuration in 2 VM, with and without SEV"*).
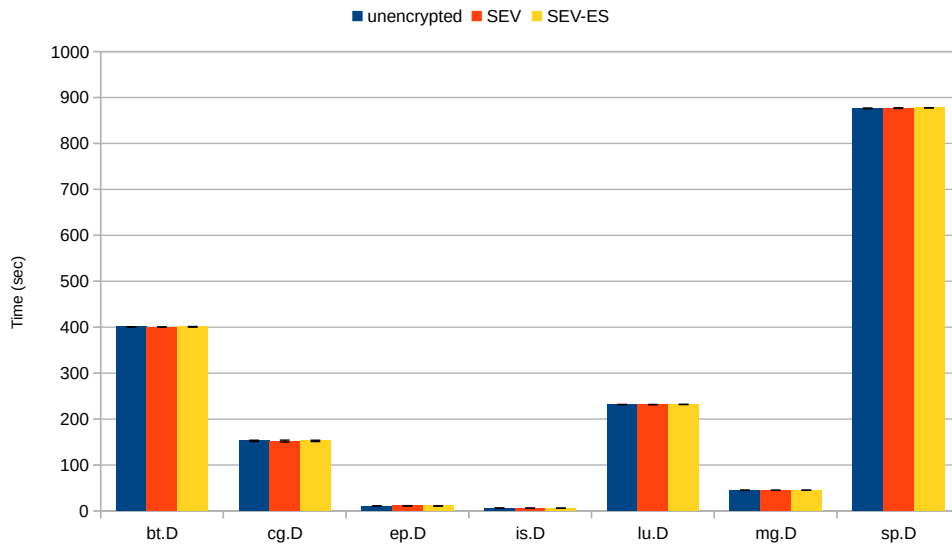


FIGURE 16: NAS COMPLETION TIME - TUNED CONFIGURATION IN 2 VM, WITH AND WITHOUT SEV

As for the memory-intensive case the overhead introduced by either of the Confidential Computing technologies is found to be really close to zero. In fact, it seems that --at least for the workloads analyzed in this document-- AMD EPYC 9005 Series Processors can handle SEV and SEV-ES with even lower overhead than its predecessors.

# 21 Conclusion

The introduction of the AMD EPYC 9005 Series Processors continues to push the boundaries of what is possible for memory and IO-bound workloads with significantly higher bandwidth and an increased number of channels. A properly configured and tuned workload can exceed the performance of many contemporary off-the-shelf solutions even when fully customized. The symmetric and balanced nature of the machine makes the task of tuning a workload considerably easier, given that each partition can have symmetric performance. And this is a property that can turn out particularly handy in virtualization, as each virtual machine can be assigned to each one of the partitions.

With SUSE Linux Enterprise 15 SP6, all the tools to monitor and tune a workload are readily available. You can extract the maximum performance and reliability running your applications on the 5th Generation AMD EPYC Processor platform.

## 22 Resources

For more information, refer to:

- AMD EPYC 9005 Series Architecture Overview (https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/user-guides/58462_amd-epyc-9005-tg-architecture-overview.pdf ↗)

- AMD Socket SP5 Power and Performance Optimization Guide for Family 1Ah Models 00h–0Fh and 10h–1Fh (https://devhub.amd.com//wp-content/uploads/Docs/58412_0.78.pdf ↗)

- Optimizing Linux for Dual-Core AMD Opteron Processors (http://www.novell.com/traininglocator/partners/amd/4622016.pdf ↗)

- Advanced Optimization and New Capabilities of GCC 12 (https://documentation.suse.com/sbp/devel-tools/html/SBP-GCC-12/index.html ↗)

- Advanced Optimization and New Capabilities of GCC 11 (https://documentation.suse.com/sbp/devel-tools/html/SBP-GCC-11/index.html ↗)

- Systems Performance: Enterprise and the Cloud by Brendan Gregg 2nd Edition(http://www.brendangregg.com/systems-performance-2nd-edition-book.html ↗)

- NASA Parallel Benchmark (https://www.nas.nasa.gov/publications/npb.html ↗)

## 23 Appendix A

Example of a VM configuration file:

```
<domain type="kvm">
  <name>vm1</name>
  <metadata>
    <libosinfo:libosinfo xmlns:libosinfo="http://libosinfo.org/xmlns/libvirt/domain/1.0">
      <libosinfo:os id="http://suse.com/sle/15.2"/>
    </libosinfo:libosinfo>
  </metadata>
  <memory unit="KiB">209715200</memory>
```

```xml
<currentMemory unit="KiB">209715200</currentMemory>
<memoryBacking>
  <hugepages>
    <page size="1048576" unit="KiB"/>
  </hugepages>
  <nosharepages/>
  <locked/>
</memoryBacking>
<vcpu placement="static">256</vcpu>
<cputune>
  <vcpupin vcpu="0" cpuset="0"/>
  <vcpupin vcpu="1" cpuset="128"/>
  <vcpupin vcpu="2" cpuset="1"/>
  <vcpupin vcpu="3" cpuset="129"/>
  <vcpupin vcpu="4" cpuset="2"/>
  <vcpupin vcpu="5" cpuset="130"/>
  <vcpupin vcpu="6" cpuset="3"/>
  <vcpupin vcpu="7" cpuset="131"/>
  <vcpupin vcpu="8" cpuset="4"/>
  <vcpupin vcpu="9" cpuset="132"/>
  <vcpupin vcpu="10" cpuset="5"/>
  <vcpupin vcpu="11" cpuset="133"/>
  <vcpupin vcpu="12" cpuset="6"/>
  <vcpupin vcpu="13" cpuset="134"/>
  <vcpupin vcpu="14" cpuset="7"/>
  <vcpupin vcpu="15" cpuset="135"/>
  <vcpupin vcpu="16" cpuset="8"/>
  <vcpupin vcpu="17" cpuset="136"/>
  <vcpupin vcpu="18" cpuset="9"/>
  <vcpupin vcpu="19" cpuset="137"/>
  <vcpupin vcpu="20" cpuset="10"/>
  <vcpupin vcpu="21" cpuset="138"/>
  <vcpupin vcpu="22" cpuset="11"/>
  <vcpupin vcpu="23" cpuset="139"/>
  <vcpupin vcpu="24" cpuset="12"/>
  <vcpupin vcpu="25" cpuset="140"/>
  <vcpupin vcpu="26" cpuset="13"/>
  <vcpupin vcpu="27" cpuset="141"/>
  <vcpupin vcpu="28" cpuset="14"/>
  <vcpupin vcpu="29" cpuset="142"/>
  <vcpupin vcpu="30" cpuset="15"/>
  <vcpupin vcpu="31" cpuset="143"/>
  <vcpupin vcpu="32" cpuset="16"/>
  <vcpupin vcpu="33" cpuset="144"/>
  <vcpupin vcpu="34" cpuset="17"/>
  <vcpupin vcpu="35" cpuset="145"/>
  <vcpupin vcpu="36" cpuset="18"/>
```

```
<vcpupin vcpu="37" cpuset="146"/>
<vcpupin vcpu="38" cpuset="19"/>
<vcpupin vcpu="39" cpuset="147"/>
<vcpupin vcpu="40" cpuset="20"/>
<vcpupin vcpu="41" cpuset="148"/>
<vcpupin vcpu="42" cpuset="21"/>
<vcpupin vcpu="43" cpuset="149"/>
<vcpupin vcpu="44" cpuset="22"/>
<vcpupin vcpu="45" cpuset="150"/>
<vcpupin vcpu="46" cpuset="23"/>
<vcpupin vcpu="47" cpuset="151"/>
<vcpupin vcpu="48" cpuset="24"/>
<vcpupin vcpu="49" cpuset="152"/>
<vcpupin vcpu="50" cpuset="25"/>
<vcpupin vcpu="51" cpuset="153"/>
<vcpupin vcpu="52" cpuset="26"/>
<vcpupin vcpu="53" cpuset="154"/>
<vcpupin vcpu="54" cpuset="27"/>
<vcpupin vcpu="55" cpuset="155"/>
<vcpupin vcpu="56" cpuset="28"/>
<vcpupin vcpu="57" cpuset="156"/>
<vcpupin vcpu="58" cpuset="29"/>
<vcpupin vcpu="59" cpuset="157"/>
<vcpupin vcpu="60" cpuset="30"/>
<vcpupin vcpu="61" cpuset="158"/>
<vcpupin vcpu="62" cpuset="31"/>
<vcpupin vcpu="63" cpuset="159"/>
<vcpupin vcpu="64" cpuset="32"/>
<vcpupin vcpu="65" cpuset="160"/>
<vcpupin vcpu="66" cpuset="33"/>
<vcpupin vcpu="67" cpuset="161"/>
<vcpupin vcpu="68" cpuset="34"/>
<vcpupin vcpu="69" cpuset="162"/>
<vcpupin vcpu="70" cpuset="35"/>
<vcpupin vcpu="71" cpuset="163"/>
<vcpupin vcpu="72" cpuset="36"/>
<vcpupin vcpu="73" cpuset="164"/>
<vcpupin vcpu="74" cpuset="37"/>
<vcpupin vcpu="75" cpuset="165"/>
<vcpupin vcpu="76" cpuset="38"/>
<vcpupin vcpu="77" cpuset="166"/>
<vcpupin vcpu="78" cpuset="39"/>
<vcpupin vcpu="79" cpuset="167"/>
<vcpupin vcpu="80" cpuset="40"/>
<vcpupin vcpu="81" cpuset="168"/>
<vcpupin vcpu="82" cpuset="41"/>
<vcpupin vcpu="83" cpuset="169"/>
```

```
<vcpupin vcpu="84" cpuset="42"/>
<vcpupin vcpu="85" cpuset="170"/>
<vcpupin vcpu="86" cpuset="43"/>
<vcpupin vcpu="87" cpuset="171"/>
<vcpupin vcpu="88" cpuset="44"/>
<vcpupin vcpu="89" cpuset="172"/>
<vcpupin vcpu="90" cpuset="45"/>
<vcpupin vcpu="91" cpuset="173"/>
<vcpupin vcpu="92" cpuset="46"/>
<vcpupin vcpu="93" cpuset="174"/>
<vcpupin vcpu="94" cpuset="47"/>
<vcpupin vcpu="95" cpuset="175"/>
<vcpupin vcpu="96" cpuset="48"/>
<vcpupin vcpu="97" cpuset="176"/>
<vcpupin vcpu="98" cpuset="49"/>
<vcpupin vcpu="99" cpuset="177"/>
<vcpupin vcpu="100" cpuset="50"/>
<vcpupin vcpu="101" cpuset="178"/>
<vcpupin vcpu="102" cpuset="51"/>
<vcpupin vcpu="103" cpuset="179"/>
<vcpupin vcpu="104" cpuset="52"/>
<vcpupin vcpu="105" cpuset="180"/>
<vcpupin vcpu="106" cpuset="53"/>
<vcpupin vcpu="107" cpuset="181"/>
<vcpupin vcpu="108" cpuset="54"/>
<vcpupin vcpu="109" cpuset="182"/>
<vcpupin vcpu="110" cpuset="55"/>
<vcpupin vcpu="111" cpuset="183"/>
<vcpupin vcpu="112" cpuset="56"/>
<vcpupin vcpu="113" cpuset="184"/>
<vcpupin vcpu="114" cpuset="57"/>
<vcpupin vcpu="115" cpuset="185"/>
<vcpupin vcpu="116" cpuset="58"/>
<vcpupin vcpu="117" cpuset="186"/>
<vcpupin vcpu="118" cpuset="59"/>
<vcpupin vcpu="119" cpuset="187"/>
<vcpupin vcpu="120" cpuset="60"/>
<vcpupin vcpu="121" cpuset="188"/>
<vcpupin vcpu="122" cpuset="61"/>
<vcpupin vcpu="123" cpuset="189"/>
<vcpupin vcpu="124" cpuset="62"/>
<vcpupin vcpu="125" cpuset="190"/>
<vcpupin vcpu="126" cpuset="63"/>
<vcpupin vcpu="127" cpuset="191"/>
<vcpupin vcpu="128" cpuset="64"/>
<vcpupin vcpu="129" cpuset="192"/>
<vcpupin vcpu="130" cpuset="65"/>
```

```
<vcpupin vcpu="131" cpuset="193"/>
<vcpupin vcpu="132" cpuset="66"/>
<vcpupin vcpu="133" cpuset="194"/>
<vcpupin vcpu="134" cpuset="67"/>
<vcpupin vcpu="135" cpuset="195"/>
<vcpupin vcpu="136" cpuset="68"/>
<vcpupin vcpu="137" cpuset="196"/>
<vcpupin vcpu="138" cpuset="69"/>
<vcpupin vcpu="139" cpuset="197"/>
<vcpupin vcpu="140" cpuset="70"/>
<vcpupin vcpu="141" cpuset="198"/>
<vcpupin vcpu="142" cpuset="71"/>
<vcpupin vcpu="143" cpuset="199"/>
<vcpupin vcpu="144" cpuset="72"/>
<vcpupin vcpu="145" cpuset="200"/>
<vcpupin vcpu="146" cpuset="73"/>
<vcpupin vcpu="147" cpuset="201"/>
<vcpupin vcpu="148" cpuset="74"/>
<vcpupin vcpu="149" cpuset="202"/>
<vcpupin vcpu="150" cpuset="75"/>
<vcpupin vcpu="151" cpuset="203"/>
<vcpupin vcpu="152" cpuset="76"/>
<vcpupin vcpu="153" cpuset="204"/>
<vcpupin vcpu="154" cpuset="77"/>
<vcpupin vcpu="155" cpuset="205"/>
<vcpupin vcpu="156" cpuset="78"/>
<vcpupin vcpu="157" cpuset="206"/>
<vcpupin vcpu="158" cpuset="79"/>
<vcpupin vcpu="159" cpuset="207"/>
<vcpupin vcpu="160" cpuset="80"/>
<vcpupin vcpu="161" cpuset="208"/>
<vcpupin vcpu="162" cpuset="81"/>
<vcpupin vcpu="163" cpuset="209"/>
<vcpupin vcpu="164" cpuset="82"/>
<vcpupin vcpu="165" cpuset="210"/>
<vcpupin vcpu="166" cpuset="83"/>
<vcpupin vcpu="167" cpuset="211"/>
<vcpupin vcpu="168" cpuset="84"/>
<vcpupin vcpu="169" cpuset="212"/>
<vcpupin vcpu="170" cpuset="85"/>
<vcpupin vcpu="171" cpuset="213"/>
<vcpupin vcpu="172" cpuset="86"/>
<vcpupin vcpu="173" cpuset="214"/>
<vcpupin vcpu="174" cpuset="87"/>
<vcpupin vcpu="175" cpuset="215"/>
<vcpupin vcpu="176" cpuset="88"/>
<vcpupin vcpu="177" cpuset="216"/>
```

```
<vcpupin vcpu="178" cpuset="89"/>
<vcpupin vcpu="179" cpuset="217"/>
<vcpupin vcpu="180" cpuset="90"/>
<vcpupin vcpu="181" cpuset="218"/>
<vcpupin vcpu="182" cpuset="91"/>
<vcpupin vcpu="183" cpuset="219"/>
<vcpupin vcpu="184" cpuset="92"/>
<vcpupin vcpu="185" cpuset="220"/>
<vcpupin vcpu="186" cpuset="93"/>
<vcpupin vcpu="187" cpuset="221"/>
<vcpupin vcpu="188" cpuset="94"/>
<vcpupin vcpu="189" cpuset="222"/>
<vcpupin vcpu="190" cpuset="95"/>
<vcpupin vcpu="191" cpuset="223"/>
<vcpupin vcpu="192" cpuset="96"/>
<vcpupin vcpu="193" cpuset="224"/>
<vcpupin vcpu="194" cpuset="97"/>
<vcpupin vcpu="195" cpuset="225"/>
<vcpupin vcpu="196" cpuset="98"/>
<vcpupin vcpu="197" cpuset="226"/>
<vcpupin vcpu="198" cpuset="99"/>
<vcpupin vcpu="199" cpuset="227"/>
<vcpupin vcpu="200" cpuset="100"/>
<vcpupin vcpu="201" cpuset="228"/>
<vcpupin vcpu="202" cpuset="101"/>
<vcpupin vcpu="203" cpuset="229"/>
<vcpupin vcpu="204" cpuset="102"/>
<vcpupin vcpu="205" cpuset="230"/>
<vcpupin vcpu="206" cpuset="103"/>
<vcpupin vcpu="207" cpuset="231"/>
<vcpupin vcpu="208" cpuset="104"/>
<vcpupin vcpu="209" cpuset="232"/>
<vcpupin vcpu="210" cpuset="105"/>
<vcpupin vcpu="211" cpuset="233"/>
<vcpupin vcpu="212" cpuset="106"/>
<vcpupin vcpu="213" cpuset="234"/>
<vcpupin vcpu="214" cpuset="107"/>
<vcpupin vcpu="215" cpuset="235"/>
<vcpupin vcpu="216" cpuset="108"/>
<vcpupin vcpu="217" cpuset="236"/>
<vcpupin vcpu="218" cpuset="109"/>
<vcpupin vcpu="219" cpuset="237"/>
<vcpupin vcpu="220" cpuset="110"/>
<vcpupin vcpu="221" cpuset="238"/>
<vcpupin vcpu="222" cpuset="111"/>
<vcpupin vcpu="223" cpuset="239"/>
<vcpupin vcpu="224" cpuset="112"/>
```

```
        <vcpupin vcpu="225" cpuset="240"/>
        <vcpupin vcpu="226" cpuset="113"/>
        <vcpupin vcpu="227" cpuset="241"/>
        <vcpupin vcpu="228" cpuset="114"/>
        <vcpupin vcpu="229" cpuset="242"/>
        <vcpupin vcpu="230" cpuset="115"/>
        <vcpupin vcpu="231" cpuset="243"/>
        <vcpupin vcpu="232" cpuset="116"/>
        <vcpupin vcpu="233" cpuset="244"/>
        <vcpupin vcpu="234" cpuset="117"/>
        <vcpupin vcpu="235" cpuset="245"/>
        <vcpupin vcpu="236" cpuset="118"/>
        <vcpupin vcpu="237" cpuset="246"/>
        <vcpupin vcpu="238" cpuset="119"/>
        <vcpupin vcpu="239" cpuset="247"/>
        <vcpupin vcpu="240" cpuset="120"/>
        <vcpupin vcpu="241" cpuset="248"/>
        <vcpupin vcpu="242" cpuset="121"/>
        <vcpupin vcpu="243" cpuset="249"/>
        <vcpupin vcpu="244" cpuset="122"/>
        <vcpupin vcpu="245" cpuset="250"/>
        <vcpupin vcpu="246" cpuset="123"/>
        <vcpupin vcpu="247" cpuset="251"/>
        <vcpupin vcpu="248" cpuset="124"/>
        <vcpupin vcpu="249" cpuset="252"/>
        <vcpupin vcpu="250" cpuset="125"/>
        <vcpupin vcpu="251" cpuset="253"/>
        <vcpupin vcpu="252" cpuset="126"/>
        <vcpupin vcpu="253" cpuset="254"/>
        <vcpupin vcpu="254" cpuset="127"/>
        <vcpupin vcpu="255" cpuset="255"/>
    </cputune>
    <numatune>
        <memory mode="strict" nodeset="0-1"/>
        <memnode cellid="0" mode="strict" nodeset="0"/>
        <memnode cellid="1" mode="strict" nodeset="1"/>
    </numatune>
    <os>
        <type arch="x86_64" machine="pc-q35-4.2">hvm</type>
        <loader readonly="yes" type="pflash">/usr/share/qemu/ovmf-x86_64-ms-4m-code.bin</loader>
        <nvram>/var/lib/libvirt/qemu/nvram/vm1_VARS.fd</nvram>
        <boot dev="hd"/>
    </os>
    <features>
        <acpi/>
        <apic/>
```

```xml
    <pae/>
    <vmport state="off"/>
    <ioapic driver="qemu"/>
</features>
<cpu mode="custom" match="exact" check="none">
  <model fallback="allow">EPYC</model>
  <topology sockets="2" cores="64" threads="2"/>
  <numa>
    <cell id="0" cpus="0-127" memory="104857600" unit="KiB">
      <distances>
        <sibling id="0" value="10"/>
        <sibling id="1" value="32"/>
      </distances>
    </cell>
    <cell id="1" cpus="128-255" memory="104857600" unit="KiB">
      <distances>
        <sibling id="0" value="32"/>
        <sibling id="1" value="10"/>
      </distances>
    </cell>
  </numa>
</cpu>
<clock offset="utc">
  <timer name="rtc" tickpolicy="catchup"/>
  <timer name="pit" tickpolicy="delay"/>
  <timer name="hpet" present="no"/>
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<pm>
  <suspend-to-mem enabled="no"/>
  <suspend-to-disk enabled="no"/>
</pm>
<devices>
  <emulator>/usr/bin/qemu-system-x86_64</emulator>
  <disk type="file" device="disk">
    <driver name="qemu" type="qcow2"/>
    <source file="/var/lib/libvirt/images/vm1.qcow2"/>
    <target dev="vda" bus="virtio"/>
    <address type="pci" domain="0x0000" bus="0x05" slot="0x00" function="0x0"/>
  </disk>
  ...
  <interface type="network">
    <mac address="52:54:00:16:1e:01"/>
    <source network="default"/>
    <model type="virtio"/>
```

```
        <rom enabled="no"/>
        <address type="pci" domain="0x0000" bus="0x01" slot="0x00" function="0x0"/>
      </interface>
      <serial type="pty">
        <target type="isa-serial" port="0">
          <model name="isa-serial"/>
        </target>
      </serial>
      <console type="pty">
        <target type="serial" port="0"/>
      </console>
      ...
      <graphics type="spice" autoport="yes">
        <listen type="address"/>
      </graphics>
      <video>
        <model type="qxl" ram="65536" vram="65536" vgamem="16384" heads="1" primary="yes"/>
        <address type="pci" domain="0x0000" bus="0x00" slot="0x01" function="0x0"/>
      </video>
      <memballoon model="virtio">
        <address type="pci" domain="0x0000" bus="0x06" slot="0x00" function="0x0"/>
      </memballoon>
      <rng model="virtio">
        <backend model="random">/dev/urandom</backend>
        <address type="pci" domain="0x0000" bus="0x08" slot="0x00" function="0x0"/>
      </rng>
      <iommu model="intel">
        <driver intremap="on" eim="on"/>
      </iommu>
      <vsock model="virtio">
        <cid auto="yes"/>
        <address type="pci" domain="0x0000" bus="0x07" slot="0x00" function="0x0"/>
      </vsock>
    </devices>
</domain>
```

# 24 Legal notice

# GNU Free Documentation License

## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

Optimizing Linux for AMD EPYC™ 9005 Series Processors with SUSE Linux Enterprise 15 SP6

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowl-edgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

Optimizing Linux for AMD EPYC™ 9005 Series Processors with SUSE Linux Enterprise 15 SP6

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/ ↗.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

## ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
 Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts". line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.